UNIVERSIDAD CARLOS III DE MADRID

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

DEPARTAMENTO DE SISTEMAS Y AUTOMÁTICA

TRABAJO DE FIN DE GRADO:

INTERFAZ DE CONTROL ICAB, EL VEHICULO AUTÓNOMO

AUTOR: DIEGO GARCÍA GARCÍA DE LEÓN

TUTOR: PABLO MARÍN PLAZA

JUNIO 2015, LEGANÉS

UNIVERSIDAD CARLOS III DE MADRID

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

El tribunal aprueba el TFG titulado "INTERFAZ DE CONTROL ICAB, EL VEHICULO AUTÓNOMO"

Tribunal:

Presidente:

RODRIGUEZ URBANO, FRANCISCO JOSE

Secretario:

FERNANDEZ HERRERO, CRISTINA

Vocal:

JARDON HUETE, ALBERTO

Suplente:

QUESADA REDONDO, ISABEL

ÍNDICE

RESUMEN

En la actualidad, se quiere controlar todo lo que nos rodea, a través de una pantalla. En consecuencia, el modelado y la creación de elementos visuales se encuentran en constante evolución, provocando la aparición de nuevas herramientas y versiones más actuales de las interfaces. A todo esto hay que sumar cómo el ser humano interactúa cada vez más con los dispositivos electrónicos —móviles, robots, control de fábricas, video vigilancia, etc.—, necesitando para ello un *software* que permita de la forma más intuitiva esta interacción humano-máquina.

El *software* y el *hardware* van de la mano en la mayoría de los proyectos de sistemas electrónicos, dado que la información proveniente de los sensores puede ser extraída y transformada mediante el *software* en una interfaz gráfica que sirva para analizar los datos y actuar en consecuencia.

El proyecto llevado a cabo a lo largo de este trabajo ha permitido obtener una interfaz de visualización de datos de la plataforma de investigación iCab —*encoders*, temperatura, control de errores, estado de los motores, *heartbit*, posición, ángulo de giro, etc.—, así como la posibilidad de controlarlo a través de una pantalla táctil.

La comunicación llevada a cabo a través de la pantalla y el vehículo se ha realizado mediante ROS —*Robot Operating System*— y el desarrollo de la interfaz gráfica, con Qt Creator. Mediante estas dos herramientas, se ha podido recibir la información recopilada por el microprocesador —cerebro de la capa a bajo nivel del vehículo—.

Los resultados obtenidos a la hora de realizar pruebas han sido buenos, aunque, debido a que este proyecto todavía se encuentra en desarrollo, no se han podido explotar al 100% las posibilidades que presenta esta interfaz.

Palabras clave: *Interfaz Gráfica de Usuario, vehículo autónomo, iCab, sistemas de control, software libre, arquitectura ROS, Qt Creator*.

ABSTRACT

Nowadays, everybody wants to control what surrounds us by means of a screen. Consequently, the processes of modelling and creation of visual elements are in constant evolution, which causes the advent of new tools and updated versions of the already existent interfaces. Added to all this, we find more than ever how humans interact with electronic devices —mobiles, robots, factory controls, video surveillance systems, etc.—, which increases the need for new software able to offer an improved intuitive medium in the interaction between machines and humans.

In addition, the software and hardware belong together in most of the projects dedicated to electronic systems, given that the information from sensors can be extracted and processed by the software in a graphical interface that serves to analyze the data and act accordingly.

The present project demonstrates how to obtain an interface of data visualization for the research platform iCab —*encoder*s, temperature, error handling, engine status, heartbit, position, rotation angle, etc.—, and the possibility to control it by means of a touchscreen.

The communication established between the screen and the vehicle has been carried out by the use of ROS —Robot Operating System— and the development of the graphical interface with Qt Creator. Using these two tools, it has been possible to receive the information collected by the microprocessor —brain layer at low level of vehicle.

The results obtained on testing have been positive, but due to the fact that this project is still developing, it has not been possible to make use of the full potential that this interface presents.

Keywords— Graphic User Interface, autonomous vehicle, iCab, control systems, open source, ROS architecture, Qt Creator.

CAPÍTULO 1. INTRODUCCIÓN

Los programas de control de sistemas son cada día más sofisticados en la industria actual. Esto se debe a la necesidad de realizar un estudio exhaustivo de la información que se recoge y poder analizar estos datos con el fin de enviar las señales precisas.

Actualmente, el principal objetivo del desarrollo de estas aplicaciones es facilitar el manejo de sistemas complejos. Para lograr este propósito, se busca automatizar el sistema de tal modo que el usuario, sin tener un amplio conocimiento sobre este, pueda manejarlo simplemente pulsando botones o introduciendo datos a través de una pantalla. Para hacer esto posible, se ha de tener un amplio número de consideraciones en cuenta para que el usuario no pueda efectuar operaciones indebidas en un momento determinado. Este último punto se lleva a cabo mediante la aplicación de una correcta lógica del sistema, como puede ser impedir el avance frontal de un vehículo cuando este se encuentra a una distancia peligrosa de los obstáculos.

Hasta hace unos años, no existían vehículos móviles —dotados de inteligencia— que fueran capaces de corregir los fallos que cometía el conductor del vehículo, que provocaban averías o accidentes, en algunos casos, mortales. Esta es la causa por la que hoy en día se busca desarrollar sistemas automatizados que cumplan con rigurosas medidas de seguridad, con la finalidad de evitar los accidentes que sucedan por fallo humano.

1.1. Objetivo

El objetivo principal de este trabajo es desarrollar una interfaz gráfica para el control del vehículo móvil automatizado iCab, con el fin de que este pueda ser manejado a través de una pantalla y puedan ser analizados los datos recogidos para efectuar modificaciones en la marcha del vehículo.

Desde la interfaz ha de ser posible activar/desactivar los controles de dirección y tracción del vehículo móvil, así como controlar el movimiento que efectúa el vehículo de forma manual o automática, teniendo en cuenta todas las restricciones necesarias para evitar las colisiones y el mal funcionamiento mediante la lógica del sistema.

Los datos que se recogen del vehículo tienen que mostrarse por la pantalla en diversos formatos: tablas de valores, *leds* de estado del sistema y marcadores de aguja para los datos de mayor interés.

Deberá conectarse esta interfaz con los servicios de ROS (*Robot Operational System*) [1], pudiendo así mantener la comunicación con el vehículo para enviar órdenes y recibir datos sobre el estado del sistema.

CAPÍTULO 2. ESTADO DEL ARTE

2.1. Interfaz gráfica

La interfaz gráfica se define en el *Diccionario de la lengua española* de la RAE [2] como «conexión física y funcional entre dos aparatos o sistemas independientes».

La interfaz gráfica es un elemento de comunicación entre el usuario y la máquina. Esta interacción entre la persona y el ordenador, es también conocida como IPO/HCI (Interacción Persona Ordenador o, en inglés, *Human Computer Interaction*)

2.1.1. Historia y evolución

A lo largo del último siglo, surgen avances en el desarrollo tecnológico que llevarán a plantear la necesidad de implementar sistemas que permitan al usuario de los nuevos equipos —cada vez más sofisticados— comunicarse con estos, leer sus datos y poder interactuar o tomar decisiones a través de la interfaz. Si bien han existido antecedentes "mecánicos" con expresión gráfica (por ejemplo, el sistema de comunicación entre el puente de mando y las calderas en los barcos), estos solo se pueden considerar elementos de comunicación que repetían la orden, más que dispositivos de interacción, ya que era unidireccional.

Uno de los primeros equipos que podrían considerarse provistos de una interfaz gráfica fue el radar, ya que representaba a través de una pantalla la localización de un elemento; sin embargo, no era una verdadera interfaz que permitiera la comunicación bidireccional humano-máquina.

Figura 2. 1: *Radar utilizado durante la Segunda Guerra Mundial.*

En la evolución de la interfaz gráfica, se pueden considerar cuatro etapas:

1. Hipótesis iniciales sobre la creación de una interfaz gráfica (1945-1973).
2. Interfaz gráfica de usuario inicial y desarrollo de la funcionalidad principal del sistema (1973-1995).
3. Personalización y estética de la interfaz gráfica (1995-2009).
4. Interfaz gráfica conectada al entorno (2010-actualidad).

2.1.1.1. Hipótesis iniciales sobre la creación de una interfaz gráfica

Con anterioridad al desarrollo de las interfaces para la comunicación entre humano y ordenador, **Bush** (1945) desarrolló su tesis, *As We may think* [3], en la cual hablaba sobre el almacenamiento del conocimiento y planteaba cómo debía aplicarse su sistema, llamado MemEx (Memory Extension). Además, desarrolló teóricamente el concepto de "ordenador personal", así como el de "hypertext" como modelo de información interconectada.

Durante la década de los 60, varios investigadores lanzaron hipótesis sobre cómo debía ser la interfaz estándar del futuro, mientras que, durante este

periodo, la relación entre los ordenadores y los humanos aún se establecía a través de tarjetas perforadas e impresoras línea a línea, situación que cambiaría durante esa década. Uno de los primeros en hablar sobre el funcionamiento de los ordenadores fue **Licklider** (1960), el cual afirmó que la problemática no era crear ordenadores productores de respuestas, sino ordenadores que se anticiparan y participasen en la formulación de preguntas [4].

En 1962, el propio **Licklider** publicó junto a **Clark** un artículo titulado *On-Line Man-Computer Communication* [5], en el cual estos investigadores exponían una lista con diez problemas por resolver para facilitar la interacción persona-ordenador. De estos, los cinco primeros debían resolverse a corto plazo; el sexto, en un tiempo intermedio y los restantes, a largo plazo:

1. Tiempo de uso compartido entre muchos usuarios.
2. Sistemas de entrada y salida para efectuar la comunicación a través de datos gráficos y simbólicos.
3. Un sistema interactivo que procese de las operaciones en tiempo real.
4. Sistemas de almacenamiento de información que permitan que esta sea recuperada.
5. Sistemas que faciliten la cooperación entre personas, con el objetivo de diseñar y programar grandes sistemas.
6. Sistemas de reconocimiento de voz, de escritura a máquina y escritura manual.
7. Comprensión del lenguaje natural, analizando el componente sintáctico y semántico del mismo.
8. Reconocimiento de las palabras en una conversación oral de varios usuarios aleatorios.
9. Descubrimiento, desarrollo y simplificación de una teoría de algoritmos.

10. Programación heurística o mediante principios generales.

Con el tiempo, varias de las observaciones realizadas se han convertido en realidad; en cambio, otras propuestas a largo plazo aún no han sido resueltas.

En 1964 **Engelbart** construyó el primer prototipo de ratón, que se movía a lo largo del eje de coordenadas X-Y de la pantalla. Además, en su tesis —*Augmenting Human Intellect: A Conceptual Framework* [6]— retomaba la idea de **Bush**, desarrollando el concepto de aumento de las capacidades cognitivas del intelecto humano a través de sistemas de almacenamiento de datos.

Este elemento, junto con el teclado y otras creaciones, constituyeron un gran avance en la interacción humano-ordenador, ya que se incorporaron elementos de entrada y salida comunes a los de la actualidad (teclado, pantalla, puntero gráfico, impresora de documentos, etc.).

En 1971, **Hansen** enumeró una serie de principios para diseñar sistemas interactivos [7]:

1. Conocer al usuario.
2. Minimizar la memorización: ítems, en vez de entrada de datos; nombres, en vez de números; comportamiento predecible y un acceso rápido a la información práctica del sistema.
3. Optimización de operaciones: rápida ejecución de operaciones comunes, consistencia de la interfaz y estructuración de la información en función del uso del sistema.
4. Mensajes de error más inteligibles: diseños que eviten errores comunes y que permitan deshacer acciones, garantizando la integridad del sistema en caso de fallo en el *software* o *hardware*.

A día de hoy, estos principios no siempre son seguidos por los creadores de aplicaciones, lo que conduce a la confusión del usuario ante mensajes de difícil comprensión.

2.1.1.2. Interfaz gráfica de usuario inicial y desarrollo de la funcionalidad principal del sistema

La interfaz gráfica comenzó a tomar importancia en el mundo de la informática a partir de la década de los 70 estimulada por la investigación y sus aplicaciones en el mundo de los negocios. Se inició la búsqueda de un método más sencillo y fácil de entender, que trabajar mediante la línea de comandos del sistema. Durante esta década el centro de investigación **PARC** (Palo Alto Reseach Center) logró crear un sistema operativo, con cierta interacción amigable ("friendly") entre las personas y los ordenadores [8].

En 1973, se desarrolló el primer sistema informático dotado de interfaz gráfica, el *Xerox Alto***,** sobre la que se podía interaccionar con el ratón sobre botones textuales [9].

Figura 2. 2: *Xerox Alto (1973).*

En 1981, los investigadores lograrían crear el ordenador *Xerox Star 8010*, que incluía todas las mejoras conseguidas a lo largo de una década de estudios. Este ordenador fue una revolución para el trabajo de oficina, ya que era fácil de usar, capaz de automatizarse para las tareas administrativas. Es en este modelo en el que aparece por primera vez el concepto de "escritorio", y que fue desarrollado siguiendo el concepto WYSIWYG (*What You See Is What You Get*), acrónimo que en inglés

significa "Lo que ves es lo que obtienes" [10]. Se trata de crear en la pantalla un resultado lo más parecido, a la impresión del documento real.

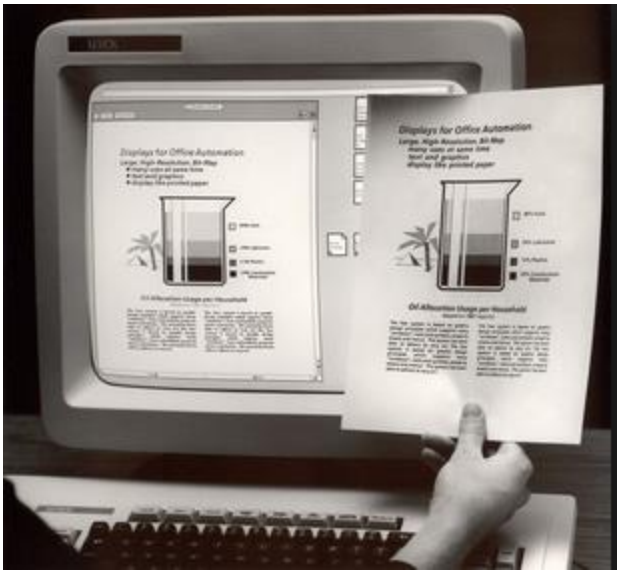


Figura 2. 3: *Xerox Star 8010, concepto WYSIWYG.*

Este modelo ya disponía de ciertas características similares a las de hoy en día como son: el manejo de ratón y el teclado; varias ventanas simultáneas; ficheros con iconos característicos.

Durante la década de los 80 y la primera mitad de los 90, se produjo una gran revolución en la industria informática con la implementación en los hogares y oficinas de los ordenadores personales (PC, "Personal Computer").

La incorporación de los ordenadores en la sociedad está totalmente ligado al uso y evolución de las interfaces gráficas, ya que estas han posibilitado a los usuarios, no especializados en el manejo y la comprensión, el uso de los ordenadores de una manera más sencilla.

En este periodo se amplían y definen los modelos vigentes del IPO/HCI de la interfaz gráfica. Los dos grandes sistemas operativos del mercado de los ordenadores, Apple y Windows, definirán el modelo del sistema operativo, desarrollando modelos de interfaces que llamen la atención al usuario a fin de lograr un mayor número de ventas.

Apple sacó en 1983 su primer modelo con interfaz gráfica de usuario integrada (GUI, "Graphic User Interface"), el *Apple Lisa* [11], orientado al

mercado del trabajo de oficina en grandes empresas. Este equipo poseía un *hardware* avanzado, memoria protegida, capacidad de multitarea hasta 2 Mb de RAM, protector de pantalla, etc. Sin embargo, no tuvo mucho éxito, debido al alto coste (9,995 dólares de la época, actualmente más de 20.000 dólares), y la competencia de IBM, apostando posteriormente por el mercado informático personal con el *Apple Macintosh* (Enero de 1984).

Figura 2. 4: *Apple Lisa (izda.) y Apple Macintosh (dcha.).*

Durante esta década desaparecieron ordenadores, interfaces y sistemas operativos de compañías como Commodore, Amiga, Next Computer o Be [12], que aportaron un granito de arena a la evolución de la interfaz gráfica de usuario. Los que realizaron un gran desarrollo durante este periodo fueron además de Apple y Windows, marcas como IBM y el proyecto de *software* libre GNU/LINUX.

El 20 de noviembre de 1985, nació del trabajo de Microsoft sobre ordenadores IBM el *Windows 1.0* [13], que fue una adaptación del sistema operativo MS-DOS a una interfaz gráfica de usuario, permitiendo desplazarse entre pantallas o "ventanas"(de ahí su nombre)haciendo clic con el ratón. Disponía de menús desplegables, barras de desplazamiento, iconos y cuadros de dialogo que facilitaban el aprendizaje. Este sistema operativo solo ocupaba 256 Kb (dos unidades de disquete de doble cara).

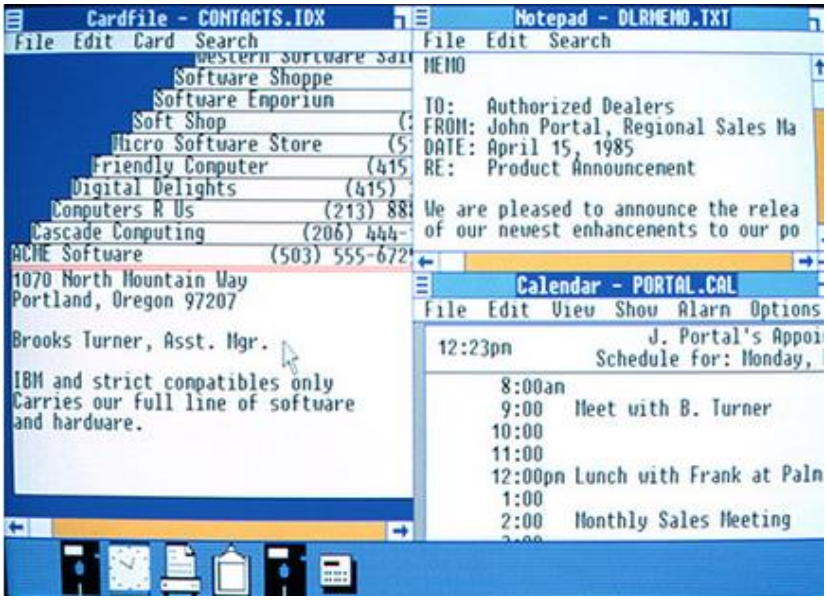


Figura 2. 5: *Interfaz del Windows 1.0*

En 1986 apareció *GEOS*, una interfaz gráfica que diseñada para el Commodore 64, posteriormente se adaptaría para IBM (figura 2.6). Esta interfaz no era controlada mediante ratón, sino por joystick en su lugar y un teclado. Permitía posicionar iconos de aplicaciones en el escritorio, imprimir, acceder a dispositivos de almacenamiento y papelera de reciclaje. No se podía redimensionar. Usaba el sistema de "scroll" (desplazamiento) para navegar por las ventanas, y poseía además un menú fijo, en la parte superior, para los comandos globales del sistema.

Figura 2. 6: *Interfaz GEOS para el Commodore 64*

En 1987 **Arcon Computers** desarrolla una interfaz llamada *Arthur* [14], diseñada a color, con un ratón de tres botones, una barra de tareas (que permite el acceso a las aplicaciones que estén activas), y un navegador de archivos parecido al del MAC OS (Operative System). Las acciones se sitúan en menús contextuales, que surgen en el contexto de una aplicación o en función del icono, activados con el botón central del ratón (actualmente clic derecho), en vez de los menús fijos situados en el escritorio.

En 1989 salió la interfaz *NeXTSTEP 1.0 GUI*, de la empresa Next Computers, que se basó en los desarrollos realizados por la empresa Adobe. En esta interfaz, los iconos eran de 48x48 pixeles, soportaba colores [15], y disponía de scrolls para la navegación. Esta interfaz incluía una barra especial, situada en la esquina derecha superior, que mostraba los programas más usados en forma de iconos, llamado *Dock* [16].

En 1990 salió el *Windows 3.0,* que dos años más tarde sacaría la versión 3.1, y juntos llegaron a vender 10 millones de copias en los dos primeros años, convirtiéndose en el más utilizado. Con la memoria virtual mejoraba los gráficos visuales. Windows comenzaba a tomar el aspecto de las versiones posteriores, con un rendimiento mejorado, a 16 colores y con iconos de mejor calidad.

En 1991 salió al mercado el *AMIGA WORKBENCH 2.04* [17], en el que se añadió un esquema de colores y el aspecto 3D fue introducido (figura 2.7). El escritorio podía dividirse verticalmente en ventanas de distinta resolución y profundidades de color. La resolución por defecto era 640x256, aunque soportaba resoluciones más elevadas.

Figura 2. 7: *Amiga Workbench 2.04, bordeado 3D.*

En 1992 IBM sacó el primer sistema operativo que poseía una interfaz que cumplía con el test de calidad internacional y con un diseño orientado a objetos, el *OS/2 2.0*. Cada archivo y carpeta eran objetos que se podían asociar entre sí y con aplicaciones. Disponía de la funcionalidad de coger los elementos, arrastrarlos y depositarlos en el "Shredder" (trituradora), que elimina los archivos.

2.1.1.3. Personalización y estética de la interfaz gráfica

A partir de 1995 llegaría la gran revolución, con la aparición del *Windows 95* [13], del que se vendieron 7 millones de copias en tan solo cinco semanas, que disponía de una interfaz orientada a objetos, que permitía abrir documentos directamente desde el escritorio, mientras que antes era necesario lanzar la aplicación antes de abrir el documento. Este cambio produce una ampliación del paradigma *WIMP* (Windows, Icons, Menus and Pointers), dónde el escritorio pasó de representar aplicaciones, minimizadas o activas, a representar iconos que pueden ser manipulados, agrupados, asociados y activados.

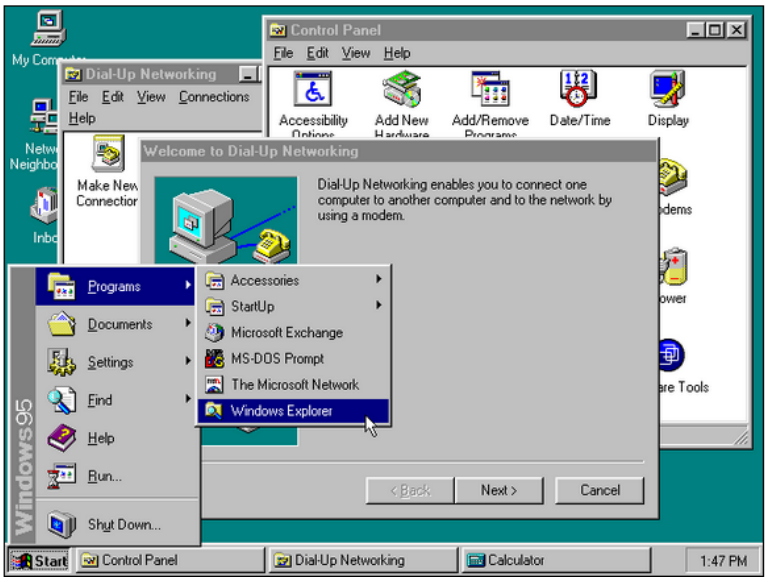


Figura 2. 8: *Windows 95, menú de inicio en forma de árbol.*

Otra novedad del Windows 95 fue la aparición del *menú de inicio*, que despliega en forma de árbol (figura 2.8) la mayoría de las aplicaciones, archivos y funciones del sistema. Este aspecto se mantiene hoy en día por su gran utilidad. Se incluyó la posibilidad de personalizar el escritorio, así como de algunas variables de la apariencia del sistema, permitiendo al usuario personalizar el ordenador a su gusto.

La interfaz gráfica había ido mejorando su capacidad de interacción, de modo que el desarrollo posterior se centró en los aspectos estéticos que demandaba el mercado. En los años sucesivos la evolución de la interfaz gráfica ya no trataba solamente de buscar una funcionalidad básica, si no que debía satisfacer dos nuevos aspectos a la hora de desarrollarla:

- La interfaz como **superficie inteligente** que añade procesos que permiten a la propia interfaz decidir su propia forma, estructurando y organizando los elementos propios. Se podría decir que la interfaz es ahora un autómata inteligente, que nos ayuda a tomar decisiones.

- **Personalización**: la interfaz había pasado de ser un mero objeto de uso, para convertirse en un producto de consumo estético, ya que al permitir al usuario cambiar ciertos aspectos de la apariencia, este puede personalizarla y expresar sus preferencias estéticas. Los primeros aspectos personalizables fueron: añadir imágenes de fondo

de escritorio, cambiar el aspecto de los iconos, ordenar elementos en un menú, entre otras. . La personalización es un apartado clave a la hora de convertir una interfaz básica, en una más personal, "amigable" (friendly), siendo más agradable y útil para el usuario.

Tras la aparición del *Windows 95*, vendrían sucesivas versiones de Windows. En 1999, fruto del proyecto *GNOME*,[18] apareció una interfaz gráfica basada en la plataforma de *software* libre GNU/Linux, diseñado en torno al concepto de escritorio informático, con un manejo de ventanas, aplicaciones y archivos, similar al de los sistemas operativos contemporáneos. Disponía de menú de inicio, con accesos rápidos a programas y ubicaciones de archivos, una barra de tareas y un área de notificación de aplicaciones, ejecutándose en segundo plano. Todas estas características podían ser modificadas, reemplazadas o eliminadas por el usuario.

Ya en el siglo XXI, Apple sacó al mercado el *MAC OS X* (2001), con un nuevo entorno gráfico llamado *AQUA* [19]**,** basado en Unix e inspirado en el agua del mar, con gotas de agua como botones y elementos (figura 2.9), usando efectos de transparencia y reflexión. Incluía el uso del Dock (usado previamente por *NeXTSTEP*), facilitando la navegación entre aplicaciones. La apariencia de esta interfaz supuso una revolución, e introdujo un gran número de cambios, a los que los usuarios tuvieron que acostumbrarse, ya que Apple continuó en esta línea en las versiones posteriores de MAC OS X.

Figura 2. 9: *AQUA GUI, MAC OS X.*

Ese mismo año su competidor Microsoft sacó el *Windows XP*, con un centro de ayuda y de soporte técnico unificado. Tenía una navegación más intuitiva por el tipo de menú de Inicio, por el orden de las aplicaciones por frecuencia de uso, por su nueva barra de tareas, por la posibilidad de agrupar ventanas abiertas de la misma aplicación, y por la accesibilidad a su panel de control. Incluía una ayuda inteligente, a través de personajes (mascotas), que daban consejos al usuario mientras se realizaban acciones sobre el sistema.

En 2007 Microsoft saco el *Windows Vista* [13] al mercado, con una interfaz modernizada, con cambios en la barra de tareas y los bordes de las ventanas (con un efecto cristalino), un aspecto que incorporaba elementos en 3D y animaciones visuales.

Figura 2. 10: *Windows Vista, animación 3D.*

En ese mismo año Apple saco el *MAC OS X LEOPARD* [20], seguía el estilo de la interfaz de Aqua, pero con algunas modificaciones:

- Barra de menú semitransparente y con bordes ligeramente redondeados.
- Un escritorio sin iconos, quedando destinado a ser el marco de fotos digitales integrado con iPhoto. En compensación, aparece la función de "Stacks" (pilas), que permitía desplegar al hacer clic sobre un elemento, los grupos de documentos y aplicaciones superpuestas,

formando una columna arqueada, o filas y columnas si estos eran demasiados. Estos elementos eran organizados por el usuario.

- La ventana que se encontraba activa, mostraba una sombra de mayor intensidad para resaltarla.

La interfaz gráfica empezó a evolucionar para adaptarse a la explosión de los nuevos dispositivos táctiles (smartphones, tabletas), precisando una interfaz que soporte e incorpore nuevas utilidades para ofrecer una funcionalidad plena.

2.1.1.4. Interfaz gráfica conectada al entorno

Nuevas necesidades vinculadas al desarrollo de las redes wifi motivan la aparición de *Windows 7* (2009) [13], creado para la era inalámbrica, donde los equipos portátiles empiezan a superar a los de escritorio, necesitando trabajar la mayor parte del tiempos conectados a redes inalámbricas. Este sistema operativo incorpora nuevas formas de trabajar con las ventanas (ajustar, inspeccionar y agitar), mejorando la funcionalidad y haciendo más divertido su uso. Aparece *Windows Touch***,** que es la tecnología táctil que permite al usuario navegar por internet, ver fotos, y abrir documentos sobre pantallas táctiles.

Apple seguirá actualizando las versiones de su sistema operativo MAC, y en 2010 sacaría el *MAC OS X LION* (v10.7), dónde Apple introducirá conceptos y funcionalidades del sistema operativo para móviles iOS, sustituyendo por ejemplo el sistema de "scrolling" anterior, añadiendo uno similar al de iOS. El MacBook ya disponía de un "touchpad" (ratón táctil de los portátiles) con una amplia gama de funciones en relación con la disposición y movimiento de los dedos sobre este, permitiendo hacer clic, desplazarse en todas las direcciones (scroll), avanzar o retroceder páginas, girar y hacer zoom.

Figura 2. 11: *MAC OS LION, iconos de aplicaciones integrados.*

A inicios de la década actual, con la evolución de los smartphones (teléfonos inteligentes) y su elevado número de ventas, el desarrollo de interfaces se centró en el desarrollo y la mejora de estos dispositivos, convirtiéndose en el principal objetivo de las compañías en detrimento de los ordenadores. Apple con el *MAC OS X LION* (figura 2,11), trata de acercar ambas interfaces a una única que permita la conexión de distintos dispositivos entre sí, transfiriendo archivos y configuraciones, para trabajar sobre varios dispositivos de manera simultánea, como si se tratase del mismo.

Los sistemas operativos para smartphones se centran en la funcionalidad táctil. Los más importantes son iOS de Apple, Windows de Microsoft, y Android de Google. Este último está basado en Linux, es de código abierto y es el más utilizado en las tabletas y smartphones de la actualidad, los nombres de sus versiones están relacionados con los dulces (Gingerbread, Ice Cream Sandwich, Jelly Bean…)

En la actualidad Microsoft con su versión del *Windows 8.1* (2013) [13], pone en el mercado una interfaz de usuario avanzada, con total funcionalidad táctil, con una nueva barra de tareas y un sistema de administración de archivos optimizado. La pantalla de inicio de *Windows 8* posee iconos que conectan con las personas, archivos, apps y sitios web (figura 2.12). Como las apps tienen gran importancia en la actualidad, se incorpora un portal de

descarga "Tienda Windows", que es similar al "App Store" de Apple o al "Play Store" de Google.

Figura 2. 12: *Windows 8.1 y su menú de inicio.*

Este sistema, altamente configurable por el usuario, utiliza la última tecnología, como es la "nube" ("cloud") que consiste en almacenar un gran volumen de datos fuera del sistema, en la "red", así como aplicaciones a las que puede conectarse a través de internet.

Apple en su última versión *MAC OS X Yosemite* (2014) [21], presenta un diseño inspirado en el *iOS 7*, pero manteniendo la metáfora de escritorio de *OS X*. Con unas altas prestaciones de continuidad, para integrar los distintos sistemas*, iOS* o *iCloud*, permitiendo así realizar operaciones entre dispositivos, como puede ser efectuar una llamada de teléfono con el iPhone desde el MAC. El sistema *Handoff* permite empezar actividades entre dispositivos, y enviar/recibir SMS y MMS desde el propio ordenador.

Dentro de las plataformas de *software* libre encontramos *Linux*, basado en Unix. Distribuido bajo la *GNU General Public License,* siempre accesible y desarrollado por multitud de programadores. Uno de los modelos de Linux más utilizados hoy en día es *Ubuntu* [22], nombre que proviene del concepto "zulú y xhosa", que significa "humanidad hacia otros". Está basado en *Debian GNU/Linux* y tuvo su primera aparición en octubre de 2004. La

última versión de este sistema es *Ubuntu 15.04*, no dispone de las características de los dos grandes del mercado, pero ofrece un alto rendimiento del sistema. Además Ubuntu se encuentra disponible para distintas plataformas (smartphones, tablets, PC's).

En el desarrollo evolutivo de las interfaces han ido apareciendo nuevas retos y problemáticas que han estimulado la investigación y desarrollo de estas haciendo que este interesante campo no se estanque y continúe evolucionando día tras día. En la actualidad, las interfaces son mucho más intuitivas y amigables gracias al trabajo previo realizado, avanzando de cara al futuro hacia una interacción humano-máquina de un modo más "natural".

2.1.1.5. Evolución de la interfaz en vehículos móviles

Los vehículos primitivos presentaban un funcionamiento principalmente mecánico, y no estaban dotados de ningún dispositivo electrónico inteligente que ayudase al conductor a conocer el estado de su vehículo. Hasta que se transformó en electrónico con el uso de sensores, y a través de la medición se enviaba información al conductor.

En 1970, el vehículo de exploración lunar no tripulado "*Lunojod*" [23], de la URSS, constituyó un hito en la historia de los vehículos de control remoto. Durante un año se desplazó un total de 10 Km por la superficie lunar realizando experimentos programados por los científicos (Figura 2.13). Tres años más tarde alunizo la segunda versión de este vehículo *"Lunajod 2*", mejorado notablemente para conseguir sus objetivos. La carrera espacial es un motor para el desarrollo de nuevas tecnologías que posteriormente serán aplicables en el uso cotidiano.

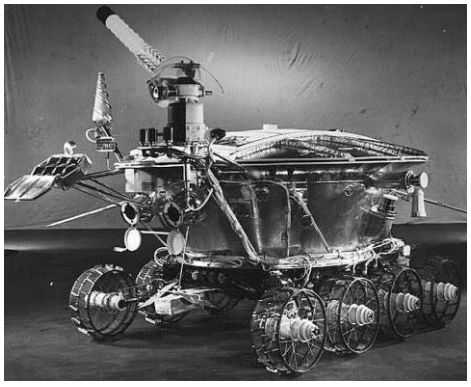


Figura 2. 13: *Lunojod I, el primer vehículo en la Luna*

Desde que en 1974 un grupo de expertos europeos identificasen 55 acciones en las que la electrónica pudiese aplicarse a los vehículos, se han hecho reales numerosas aplicaciones tales como sistema de inyección (EFI, *Electronic Fuel Inyection*), antibloqueo (ABS, *Anti-Block System*), control automático de crucero (CC, *Cruiser Control*), llegando finalmente a los sistemas de diagnóstico abordo (OBD, *On Board Diagnostics*).

Este sistema de diagnóstico a bordo apareció en los años 80 por la necesidad de los fabricantes para cumplir con legislaciones medioambientales de emisión de gases. La función del OBD es monitorizar el sistema de control de gases y adjudicarle códigos que designan fallos, almacenándolos en la memoria, llamada *Engine Control Unit*. Una señal luminosa en el panel de contol con el texto *"Check Engine*", indica al conductor la necesidad de revisar el vehículo, debido a que se ha encontrado una anomalía.

El sistema OBDII al final de los 90 aporta mejoras notables y una ampliación de funciones. Se obliga al uso de códigos de fallo universales con el sistema DLC (*Data Link Connector*) abarcando la monitorización. Además de los gases, controla todos los sensores que tiene un vehículo permitiendo una información global en tiempo real.

Actualmente la integración de los ordenadores y sistemas multimedia pone en marcha el diseño y desarrollo de sistemas más complejos, como el *Advanced Driver Assistant System* (ADAS) o el *In Vehicle Information System* (IVIS).

La aparición de los sistemas GPS en los vehículos supuso un gran avance para la navegación, sin embargo, hay ciertos detractores que consideran este dispositivo como un elemento de distracción para el conductor. Las investigaciones futuras sugieren la llegada de interfaces hápticas [24], con la integración de visión a través de una computadora, estimulación táctil y órdenes por reconocimiento de voz están siendo desarrolladas en la actualidad.

En el futuro próximo la interacción humano-computador, además de controlar los distintos elementos del vehículo, tendrá en cuenta el estado

del usuario y su entorno, asistiéndolo de manera proactiva [25]. Con esta asistencia se espera mejorar la confortabilidad del conductor y la seguridad del mismo, como puede ser en caso de detección de somnolencia en el conductor (a partir de sus constantes vitales), sugerir efectuar una parada para que descanse.

2.1.2. Interfaz del futuro

En el futuro se espera que los dispositivos se encuentren conectados al entorno que nos rodea, alimentados por la "nube" y atento a las necesidades del usuario, anticipándose a estas.

El desarrollo de las interfaces hace que sean cada vez más intuitiva y amigable la relación del usuario con la máquina, acercándolos hasta el punto de que formen una simbiosis en la que la maquina sea como una nueva extremidad. Esto lleva a que hoy día se busque el control sobre estas interfaces a través de nuestro cuerpo, con pensamientos, órdenes vocales y gestos, que interpretados por el sistema, nos escuchará, seguirá nuestra mirada y reaccionará en consecuencia.

Ya se han creado sistemas con interfaces multimodales: con *Siri*, se puede escuchar lo que el usuario quiera decir, con *Kinect*, percibimos los movimientos, y el mando de PS4, que puede percibir movimientos, dispone de panel táctil y botones configurados por el sistema. También existen dispositivos capaces de medir los impulsos eléctricos de los músculos, como es el *Myo*" (figura 2.14) y que serán traducidos en comandos por el sistema operativo.

Figura 2. 14: *Myo, control mediante los músculos.*

Un gran campo de investigación es el destinado a la medición de los impulsos enviados por el cerebro, de modo que con simplemente pensar en lo que se desea hacer, en el futuro las máquinas las transformarán en órdenes.

Otro aspecto importante, que ya se está llevando a cabo a la hora de desarrollar nuevas aplicaciones, es anticiparse a lo que el usuario le interese. En esta dirección la tecnología *BigData*, que como su nombre indica se basa en el uso de un gran almacén de datos, que analizado de forma inteligente puede dar grandes aplicaciones. El sistema operativo recopila la información sobre el usuario creando un perfil, que posteriormente transfiera información seleccionada a las aplicaciones, mediante la nube, ofreciendo un servicio personalizado. La identificación cada vez que el usuario utilice una aplicación hará que, con independencia del dispositivo en el que se encuentre, se incorporen las características propias del usuario. Actualmente Google ya incorpora a su navegador los marcadores, páginas favoritas, contraseñas y preferencias a la hora de efectuar búsquedas.

Para interactuar con la máquina, como si se estuviera frente a otro humano, esta debe disponer no solo de inteligencia artificial, sino también de una inteligencia emocional, que permita a la máquina comprender al usuario, y actuar en consecuencia.

Las medidas de las emociones, ya son posibles hoy en día, a través de variables psicofisiológicas, como el ritmo cardiaco, la conductancia de la piel, la sudoración, la dilatación de pupilas o la tensión de determinados músculos, aunque estas variables muestran una precisión limitada. La expresión facial también puede ser reconocida por sistemas actualmente en desarrollo (reconocimiento del rostro humano y de su expresión facial), aunque es fácil simular facialmente un estado de ánimo que no sea el estado real.

Un sistema operativo de hoy en día que busca unificar el uso de aplicaciones a través de la web, es el *Chrome OS* de *Google*, dónde todo gira en torno a su navegador, y cuya memoria principal se situaría en la nube, mediante *Google Drive*. Tiene ciertos defectos, tales como que no puede ejecutar programas nativos de otros sistemas operativos o que si se encuentra sin conexión a internet no puede acceder a un amplio número de aplicaciones.

Google también está invirtiendo en su producto más reciente como es *Google Glass* [26], que permite sacar fotos y grabar en video. Integra una cámara de 5 megapíxeles, un prisma en frente del ojo derecho, una superficie táctil en la patilla derecha, auriculares y micrófono. Estas "gafas" bajo un sistema operativo *Android*, conectadas a internet mediante *Wi-Fi* o *Bluetooth*, son capaces de entender órdenes vocales y leer texto.

Todos estos avances, tanto en los nuevos dispositivos como en sus interfaces alcanzarán en un futuro próximo la madurez necesaria para salir al mercado, permitiendo al usuario controlar todo su entorno a su antojo, sin necesidad de aprendizaje alguno dada la sencillez y obviedad de su manejo, idea central de su desarrollo.

2.2. DARPA Challenge: vehículos autónomos

Como se ha explicado anteriormente las interfaces tienen múltiples aplicaciones aparte de la interacción humano-máquina. Los vehículos autónomos requieren de una interfaz interna que comunique los dispositivos como sensores, GPS o cámaras con el ordenador de a bordo del vehículo, encargado de originar una respuesta a estos mensajes recibidos.

DARPA, es un equipo de investigación del departamento de defensa de Estados Unidos, cuyas siglas significan "*Defense Advanced Research Projects Agency*". ARPA *(Advanced Research Projects Agency*) se crea en 1958 teniendo como objetivo, promover la investigación tecnológica, con la finalidad de conseguir tecnologías de vanguardia para la seguridad estadounidense. En este equipo participan algunos de los investigadores de los que se ha hablado con anterioridad, como **Licklider** y **Engelbart.**

DARPA organiza eventos, para incitar y motivar a empresas y universidades a participar en ellos, consiguiendo de este modo avances en la materia sobre la que trate cada uno de ellos. De esta manera en 2002 se anunció el *DARPA Grand Challenge* (2004) [27], que consistia en una carrera de vehículos autónomos a través del desierto del Mojave (EE. UU.), con un recorrido de 142 millas, del que se disponía de un listado de puntos intermedios, y con un premio de 1 millón de dólares. En esta primera edición ningún vehículo logró terminar la carrera, siendo el que más recorrido logró realizar (7,4 millas) el *Sandstorm* del equipo *Red Team* de la Universidad Carnegie Mellon (Pittsburgh, Pensilvania) [28].

Los vehículos autónomos son aquellos vehículos capaces de navegar sin la intervención humana, mediante el uso de sensores y sistemas de posicionamiento, que determinan que hacer para alcanzar su objetivo [29].

Esta edición de 2002 sirvió de toma de contacto para todos los equipos, y pese a los malos resultados obtenidos, fue muy útil, ya que permitió ver cuáles eran los errores más comunes.

La demostración de las potenciales capacidades de estos vehículos se llevó a cabo en los eventos disputados más adelante. En 2005, se disputó una nueva edición el 8 de octubre, en la que se incrementó el premio a 2

millones de dólares. Se registraron 195 equipos, de los cuales 23 disputaron la carrera y tan solo 5 lograron terminarla. Las condiciones de la carrera fueron 175 millas en menos de 10 horas [30], con un recorrido que salía y llegaba al mismo punto, en Primm (a 30 millas al sur de Las Vegas).

El ganador fue el *Stanley* de la Universidad de Standford (figura 2.15), con un tiempo de 6 horas, 53 minutos y 58 segundos [31]. Este vehículo fue un *Volkswagen Touareg R5*. Equipado con 6 procesadores Intel, y una serie de sensores y actuadores para la conducción autónoma.

Figura 2. 15: *"Stanley", ganador del DARPA Grand Challenge*

Debido al éxito del *Grand Challenge*, en noviembre de 2007 se disputó la primera *DARPA Urban Challenge* [29], que se desarrollaba en un entorno urbano, en el que se debía tener un vehículo capaz de conducir con tráfico, y realizar maniobras complejas como aparcar, cruzar intersecciones, esquivar obstáculos, etc. Se efectuaron pruebas de clasificación para los 89 equipos inscritos inicialmente, desde requisitos técnicos y videos, hasta pruebas en las oficinas de los equipos. Finalmente fueron 35 los invitados al NQE *(National Qualification Event*), dónde se realizaron tres pruebas distintas:

1. El NQE A: consistía en circular en un circuito con tráfico, de modo que se ponía a prueba la seguridad de los vehículos. Solo un vehículo autónomo golpeo a otro vehículo.

2. El NQE B: estaba dividido en dos partes, la primera consistía en una larga recta de 2,8 millas, sobre la que tenían que circular en línea recta, con una sección de vehículos aparcados en paralelo y obstáculos de carretera que debía esquivar, y la segunda parte consistía en aparcar entre dos vehículos y desaparcar de forma segura.

3. El NQE C: consistía en intersecciones de carretera, dónde debían resolver las situaciones interpretando la preferencia y procediendo con seguridad la maniobra. Otra prueba era la de colocar bloqueos de carreteras, obligando al vehículo a realizar una maniobra de retorno y buscar una ruta alternativa.

Una vez realizado el recuento de puntuaciones de los vehículos, 11 de ellos pasaron al evento final, en el que los equipos debían recorrer una serie de puntos del mapa en un orden específico, este orden no fue entregado hasta 5 minutos antes de la prueba. Además, se incorporaron vehículos pilotados para simular el tráfico. Tras cuatro horas de prueba entró el *Standford Racing Team*, y un minuto más tarde entraría el *Tartan Racing* de la Universidad Carnegie Mellon. La clasificación final del evento tras analizar las infracciones fue la siguiente:

1. *Tartan Racing* – Pittsburgh, PA ($2,000,000)

2. *Stanford Racing Team*– Stanford, CA ($1,000,000)

3. *Victor Tango* – Blacksburg, VA ($500,000)

Los vehículos partícipes en esta edición demostraron tener un alto nivel de seguridad, ya que no sucedieron apenas incidentes, con lo que el objetivo de DARPA de conseguir un vehículo autónomo fiable se hizo realidad.

2.3. Proyecto iCab

El proyecto iCab *(Intelligent Campus Automobile*) consiste en el desarrollo de un vehículo inteligente autónomo, capaz de navegar por un entorno urbano, mediante GPS, IMU *(Inertial Measurement Unit*), el análisis de imágenes y la interpretación de un escáner láser. La implementación de este vehículo se realiza sobre un carrito de golf modelo "E-Z-GO", al que se le ha llamado "iCab vehículo autónomo".

Figura 2. 16: *Vehículo autónomo iCab 1.*

Para alcanzar este objetivo, hay que modificar el carrito para convertirlo en un vehículo autónomo. Estas modificaciones se efectúan a nivel de *hardware* y de *software*.

Respecto al *hardware* del vehículo se ha sustituido el volante de control de dirección por un sistema *motor-encoder*, que es manejado a través de un microcontrolador en una placa electrónica. También el pedal del acelerador se sustituye por un sistema de control del sistema de tracción gobernado por un microcontrolador. Además se han añadido los sistemas de visualización (ver figura 2.16), con una cámara binocular *(BumbleBee 2*), que tiene una resolución de 1032x776 pixeles a 20 imágenes por segundo [32], y el láser (SICK LMS 291), capaz de escanear un rango de 180º con una resolución de 0,25º [33]. Estos dos elementos se encuentran conectados a un ordenador embebido de a bordo, que cuenta con un sistema operativo *Ubuntu*, y posee una pantalla táctil de 7 pulgadas.

Desde el punto de vista electrónico y del *software* que posee, se puede observar una serie de bloques que componen el control del vehículo, así como la alimentación que presenta.

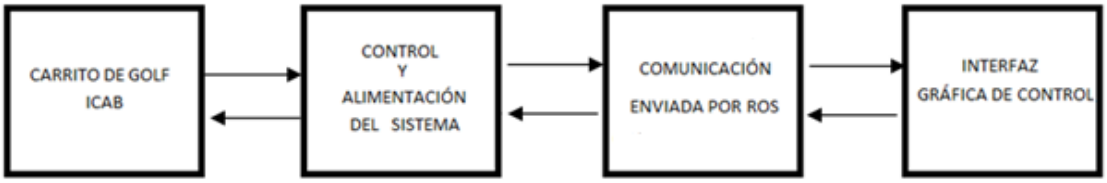


Figura 2. 17: *Bloques del proyecto iCab con la arquitectura ROS.*

Como se observa en la ilustración (figura 2.17), a la izquierda se sitúa el carrito de golf iCab, que recibe las señales de control y alimentación de los motores del vehículo y envía las señales de los sistemas de instrumentación a la placa de control y alimentación del sistema situada a continuación. Este bloque se encarga de analizar los parámetros y enviar una respuesta en función de la lectura recibida. Por otro lado la comunicación de ROS sirve de intermediario entre la interfaz gráfica de control y la placa de control, enviando y recibiendo mensajes.

Mediante ROS tambien se analizan la adquisicion de imágenes a través de la cámara, y con otro paquete se analiza el escaneado efectuado por el laser.

El vehículo tiene tres capas de control:

1. **Reactiva (bajo nivel)**: estas son las peticiones realizadas al sistema, relacionadas con el movimiento del vehículo, como avanzar, retroceder, girar a derecha o a izquierda y parar.

2. **Híbrida (secuenciador)**: genera una serie de tareas a realizar por el vehículo, dónde el éxito reside en la precisión de ejecución de las tareas simples.

3. **Deliberativa (alto nivel)**: de forma lógica el vehículo reacciona para alcanzar el objetivo (un punto del mapa), para lo que utiliza la localización (GPS), serie de pasos para alcanzar el objetivo, navegación y mapeado. Es el funcionamiendo del vehículo autonomo, que a través de sus sistemas de medida es capaz de decidir rutas, y reaccionar ante imprevistos.

Para poder llevar a cabo operaciones de reacción es necesaria una buena interfaz de control que permita al usuario poder actuar sobre el vehículo de un modo rápido, sencillo y seguro.

CAPÍTULO 3. DESCRIPCIÓN GENERAL

Como se ha mencionado en apartados anteriores, en este proyecto se propone desarrollar una interfaz gráfica de control para el vehículo autónomo iCab, que mediante la arquitectura ROS (*Robot Operating System*) mantiene la comunicación con el vehículo autónomo, permitiendo enviar órdenes de movimiento y visualizar el estado del vehículo.

A continuación, se detallaran las herramientas escogidas para la creación de la interfaz de control del vehículo, así como el funcionamiento de una arquitectura basada en ROS.

3.1. Sistema Operativo: Ubuntu 14.04

El sistema operativo de *Linux, Ubuntu 14.04*, es una de las últimas versiones (Abril 2014) de este sistema operativo de *software* libre (gratuito) y es sobre el que ROS ha sido desarrollado y sobre el que funciona de forma segura. Otros sistemas operativos sobre los que ROS puede funcionar pero que no tienen soporte oficial son OSX *(Apple*) y *Android* NDK (*Google*).

Para la ejecución de los programas sobre Ubuntu, es muy cómodo abrir una "terminal", que es el equivalente al "cmd" (línea de comandos de *Windows*) y ejecutar comandos para el manejo de aplicaciones instaladas. Las instalaciones de estas aplicaciones pueden efectuarse desde diferentes sitios:

- **Terminal**: para abrir esta ventana se puede usar el acceso directo "Ctrl+Alt+T", o buscar la aplicación en el menú de inicio de *Ubuntu*. Una vez abierto es necesario ejecutar el comando "apt-get" seguido del nombre del paquete que se desea instalar, que deberá encontrarse en alguno de los repositorios incluidos en el archivo "/etc/apt/sources.list". Si se necesita añadir un repositorio se puede editar este archivo o ejecutar el comando "add-apt-repository <repository_name>". En muchas ocasiones para que Ubuntu permita la ejecución de estos comandos es necesario escribir "sudo" antes del comando, y *Ubuntu* te pedirá la contraseña del administrador del equipo.

- ***Ubuntu Software Center***: es una aplicación desde la que se puede buscar programas, instalar algunos descargados y desinstalar aquellos que se encuentren instalados.

- ***Synaptic***: es una aplicación que hace de "front-end" (interfaz) del "apt", que permite buscar e instalar programas. Esta aplicación no viene instalada por defecto en *Ubuntu*, por eso es preciso instalarla primero.

Mediante uno de estos tres elementos se podrá instalar ROS en nuestro sistema.

3.2. Comunicación con iCab: ROS

Como se ha comentado con anterioridad, la arquitectura sobre la que está basada la comunicación entre la interfaz de control y el vehículo está desarrollada en ROS *(Robot Operational System*).

ROS es una herramienta desarrollada con el objetivo de poder crear *software* para el manejo de robots, y es que un vehículo autónomo es un robot, ya que está dotado de inteligencia artificial. Generar el comportamiento de los robots es una tarea compleja, por eso mediante ROS se busca la colaboración de los desarrolladores para poder compartir el conocimiento sobre las diferentes áreas de la robótica (navegación por mapas, navegación por visualización de imágenes…), y así facilitar la generación de robots basados en trabajos ya probados con anterioridad.

ROS se desarrolló inicialmente a mediados de la primera década del siglo XXI, mediante varios proyectos de la Universidad de Standford, como el STAIR (*STandford Artificial Inteligent Robot*) y el programa PR (Personal Robot), que crearon prototipos de *software* flexible y dinámico para el uso de robots. Desde 2007, el laboratorio de investigación robótica *Willow Garage*, generó importantes recursos para ampliar los conceptos y crear sistemas probados eficazmente. Un amplio número de investigadores contribuyeron en la principal idea de ROS y en sus paquetes de *software*. El *software* generado siempre fue de libre acceso bajo la licencia de código abierto BSD, convirtiéndose en una plataforma ampliamente usada por la comunidad robótica [1].

Como plataforma de *software* libre, un gran número de instituciones han utilizado la plataforma ROS, generando su propio código en un repositorio de ROS, con un control total sobre el mismo, eligiendo la opción de hacer que su repositorio sea público, recibiendo el reconocimiento y el crédito de sus éxitos, y por otro lado beneficiando a la plataforma, que puede mejorar el código publicado, como sucede en todos los proyectos de código abierto.

Esta sinergia promovida por las plataformas de código abierto, hace que los avances científicos se produzcan con mayor rapidez [34], lo que se traduce en una evolución de la tecnología como nunca antes había existido. Poner al alcance de todos los últimos esfuerzos realizados por las instituciones de I+D es muy positivo, ampliando el conocimiento en los aspectos que más interesan a la sociedad científica. Cada investigador puede aportar su granito de arena para mejorar un proyecto u otro, de modo que si cada uno hace una mejora sobre los proyectos, permitirá que la calidad del mismo supere las expectativas iniciales. Estas plataformas ayudan a que cualquier persona pueda recrear, sin tener mucho conocimiento de la materia, cualquier proyecto que ya haya sido probado.

La arquitectura basada en ROS consiste en un número de módulos de *software* encapsulados como los nodos, con un nodo maestro y otros funcionales [35]. ROS sirve para construir sistemas con muchos procesos, que están compuestos generalmente por los siguientes elementos particulares de ROS: nodos, mensajes, topics (temas) y servicios. [36]

Los nodos son procesos que realizan operaciones, que emiten mensajes a otros nodos, estos mensajes son estructuras de datos primitivos, que pueden combinarse. Los nodos envían el mensaje a través de una publicación, lo que provoca la generación de un topic, al que otro nodo se subscribe para recibir este mensaje. Hay un nodo maestro que controla el sistema ROS y que tiene que encontrarse en ejecución antes de poder ejecutar cualquier otro.

Los servicios son funciones que reciben un mensaje de petición y generan un mensaje de respuesta, que pueden ser estructuras de variables contenidas en estos. Estos servicios pueden ser generados o utilizados por los nodos, encargados de enviar la petición.

En la figura 3.1 se observa la aplicación de ROS para mostrar gráficamente los nodos, topics y servicios que se encuentran en ejecución, pudiendo observar que nodo se subscribe a un topic creado por otro nodo o si mismo. Este programa se llama "ROS *graph*" y se ejecuta desde la terminal, que nos abrirá una ventana a través de la que podemos elegir que elementos queremos que se muestren.

Figura 3. 1: ROS Graph, tutorial turtlesim.

En el caso de la figura, el nodo "/teleop_turtle" que está publicando el topic "/turtle1/command_velocity", al que se encuentra subscrito el nodo "/turtlesim".

El proyecto iCab reconfigure dispone de un nodo de control llamado *movement_manager*, encargado de gestionar los mensajes enviados por el nodo de la interfaz de control, llamado *icab_reconfigure*, y de enviar la información proveniente de la placa de control a través de un *topic* llamado *status_info*, al que el paquete de la interfaz se encuentra subscrito.

3.3. Herramientas de desarrollo: Qt Creator

Para generar una interfaz gráfica en C++ se ha escogido *Qt Creator*, que es un *framework* (escrito en c++) de desarrollo integrado de código (IDE, *Integrated Development Enviroment*) [37], desarrollada como *software* libre y de código abierto, y en consecuencia gratuita. Tiene un preprocesador, MOC (*Meta-Object Compiler*), encargado de analizar los archivos de C++ y generar recursos estándar, que podrán ser compilados por cualquier compilador de C++ básico, haciendo que pueda ser exportado a cualquier ordenador que posea un compilador genérico como MSVC, MinGW, Clang o GCC.

Esta herramienta fue desarrollada por **Eirik Chambe-Eng** y **Haavard Nord** en 1990, ambos de la compañía *Trolltech*, vendieron las licencias y ofrecieron soporte. Actualmente esta compañía ha pasado a llamarse *The Qt Company* y es una filial de *Digia Plc. Qt* está compuesto por un gran número de empresas y personas que colaboran con esta plataforma.

Esta herramienta se puede combinar perfectamente con ROS creando paquetes a través de *Catkin build system*, basado en el compilador *CMake*.

Qt Creator tiene una serie de características interesantes en el desarrollo de código:

- Editor de código con soporte para C++, QML y ECMAscript.
- Herramientas de navegación, resaltado de sintaxis y autocompletado.
- Control estático de código y estilo de la escritura.
- Paréntesis coincidentes y modos de selección.
- Acceso rápido a últimos proyectos y sesiones.
- Compatibilidad de proyectos en diversas plataformas, escritorio o móviles.

Qt Creator dispone de un editor de código que ayuda a los desarrolladores a mantener un estilo del código y una legibilidad:

- Sintaxis de funciones, símbolos y archivos en C++.
- Opciones de autocompletado, mostrando las opciones disponibles, como funciones y variables.
- Aplicación de tabulación automática.
- Contracción y expansión de funciones (plegado de código).
- Herramienta de navegación para seguir la implementación de una función y otra información.
- Numeración de las líneas de código, e identificación exacta de los errores en el mismo.

Con estas y otras características el desarrollo de aplicaciones a través de este framework es más sencillo, ahorrando tiempo y trabajo. Además dispone de una herramienta para el desarrollo gráfico llamada *Qt Designer*.

Otra alternativa al uso de Qt, es el Eclipse IDE, que es otra herramienta de desarrollo de código. Sin embargo, debido a la facilidad de Qt para exportar proyectos a otros sistemas, así como al conocimiento y preferencia personal sobre Qt y su interesante herramienta *Qt-designer*, se escogió esta para desarrollar la interfaz.

3.3.1. Qt Designer

Qt Creator posee una herramienta de diseño gráfico llamada *Qt-designer*, que permite diseñar rápidamente pequeñas aplicaciones visuales (widgets), que a su vez existen predefinidas dentro de la librería de Qt y pueden ser utilizadas para crear nuestra interfaz de control. Esta herramienta muestra el resultado final de la interfaz que se está creando, siguiendo el concepto de WYSIWYG *(What You See Is What You Get*), observando así el resultado final de nuestra interfaz. El funcionamiento de esta herramienta consiste en la generación de bloques de código en un archivo ".ui" con lenguaje XML (*eXtensive Markup Language*), a la vez que vamos añadiendo elementos al diseño.

Qt-designer tiene cuatro modos de edición con diversas funcionalidades:

- El modo "Edit" (editar, figura 3.2), es el modo general de uso, que sirve para cambiar la apariencia del sistema, añadiendo elementos, distribuyéndolos y variando las propiedades. Para añadir elementos al sistema simplemente hay que arrastrarlos desde la ventana de objetos situada en el margen izquierdo.

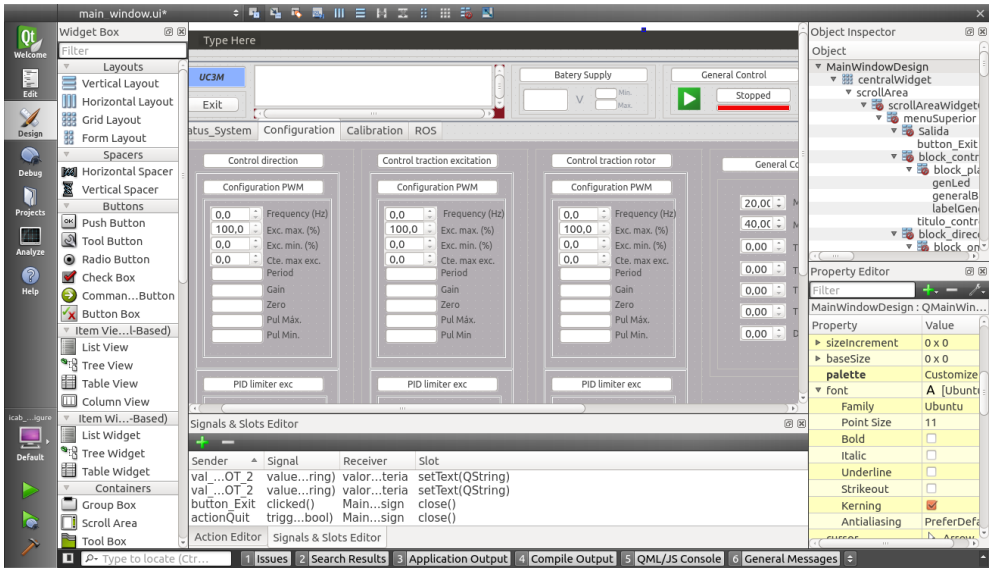


Figura 3. 2: *Edit Mode de Qt Designer.*

- El modo "*signals and slots*" (señales y conexiones, figura 3.3) permite conectar los elementos visuales, dotando al sistema de cierta inteligencia (acción/reacción), indicando el comportamiento que debe tener en determinados casos como por ejemplo, pulsar un botón ("clicked()") y que se cierre la ventana ("close()"). Este mecanismo es muy intuitivo ya que de un modo visual se conectan dos objetos arrastrando una flecha de uno a otro, a los que se les indica la operación que sucederá.

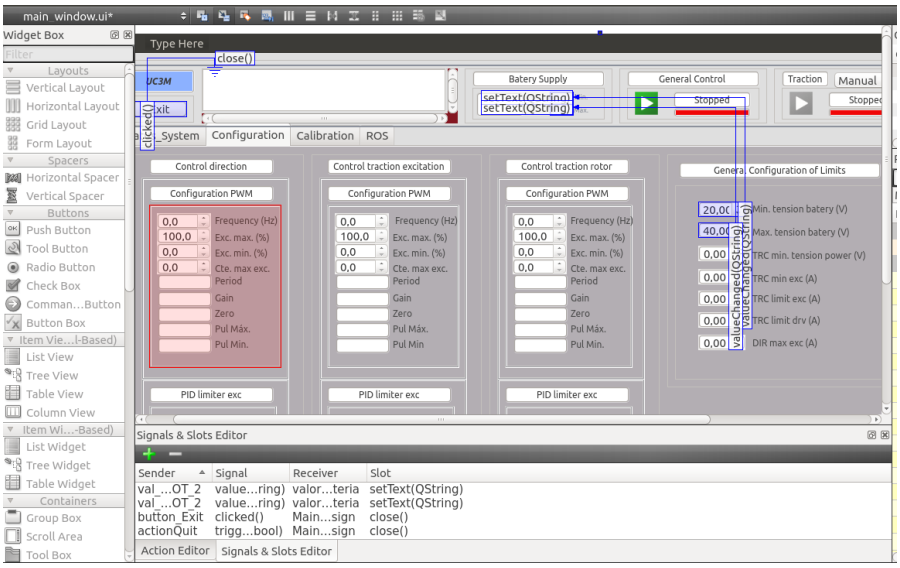


Figura 3. 3: *Signals & Slots sobre la interfaz del proyecto.*

- El modo "*Buddy Editing*", sirve para enlazar objetos con etiquetas (QLabels), sirve para identificar un elemento rápidamente, y dirigirse al mismo.

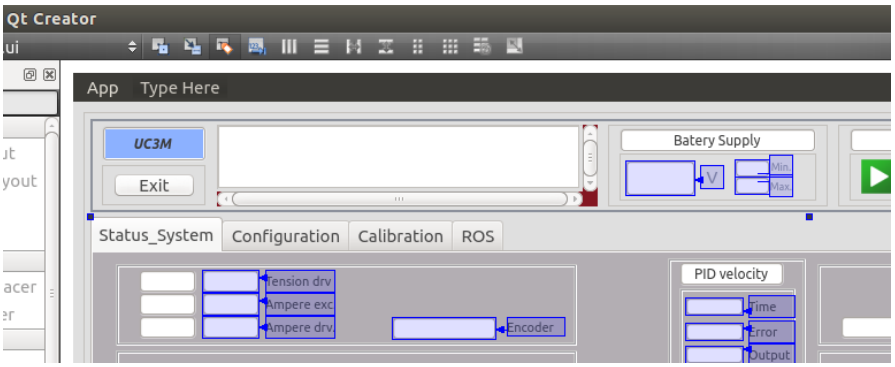


Figura 3. 4: *Buddy Editing, identificación de elementos con sus etiquetas.*

- El modo "*Tab Order*", sirve para establecer el orden en el que se sitúa el teclado sobre los distintos objetos, de modo que tabulando se seguirá este orden.

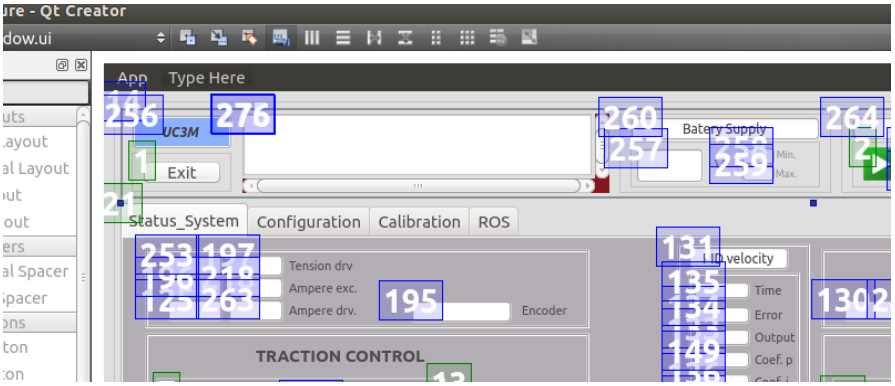


Figura 3. 5: *Orden de tabulado por la herramienta Tab Order de Qt.*

A parte de esta herramienta tiene otra llamada "*Go to Slot*", que sirve para generar la función a partir de la señal generada por un objeto. Estas señales son generadas en un archivo a parte "*MainWindow.cpp*" y "*MainWindow.h*". Los archivos ".cpp" son las clases del programa, donde se escriben los métodos de las funciones, y el ".hpp" es el archivo de cabecera, donde las funciones son declaradas.

A continuación, se van a detallar los widgets más interesantes de la herramienta, y que más adelante serán aplicadas en el proyecto. Estos elementos se encuentran divididos en el panel de Qt Designer en:

- **Layouts**, sirven para ordenar los objetos de una forma determinada, aplicando un tamaño y espaciado preestablecido. Existen cuatro tipos de layout (figura 3.6): el *Vertical Layout, Horizontal Layout, Grid Layout,* y el *Form Layout*. Facilita la colocación de los objetos arrastrados sobre la pantalla. El que se ha utilizado en el proyecto es el *Grid Layout* que permite situar elementos sobre una cuadrícula.

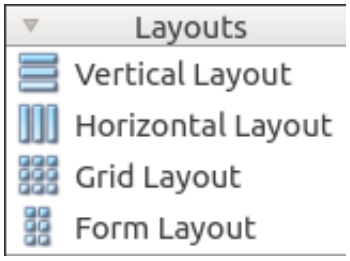


Figura 3. 6: *Layout Qt Designer*

- **Spacers** (figura 3.7)**,** sirven para mantener unos márgenes entre los objetos y el tamaño de ventana sobre el que se trabaja.

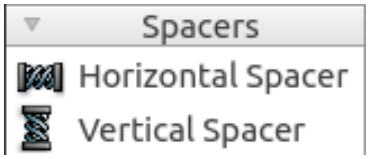


Figura 3. 7: *Spacers Qt Designer*

- **Buttons**, existen una serie de objetos sobre los que la función principal es hacer clic para transferir una orden o información determinada (figura 3.8). En este proyecto se ha utilizado el botón general, llamado *Push Button*, sobre el que se puede modificar a través de las propiedades para crear un botón a la medida de la interfaz, ya sea con texto o con imágenes. Existen otros botones predefinidos, como el botón circular *(Radio Button*) para seleccionar alguna opción, otro con una herramienta como imagen llamado *Tool Button* y otro que se llama "Check Box" que sirve para marcar la casilla o demarcarla.

Figura 3. 8: *Buttons Qt Designer*

- **Item Views**, sirven para mostrar una serie de datos en diversos formatos.

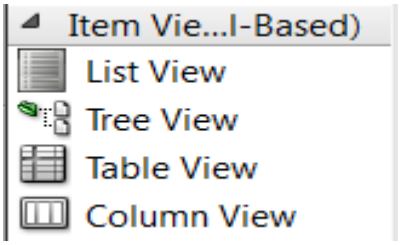


Figura 3. 9: *Items View Qt Designer*

- **Containers**, como su nombre indica son contenedores sobre los que se depositan otros objetos, tienen distintas formas y propiedades (figura 3.10). Si como el *Scroll Area* aporta barras de desplazamiento para el área seleccionada, el *Tab Widget* ofrece distintas pestañas sobre las que añadir elementos, el *Frame* sirve principalmente para agrupar los objetos y actuar sobre ellos a través de este y además tiene funcionalidad visual. Otros, como el *Dock Widget* y el *MdiArea*, sirven para superponer sobre esta elementos, como si fuesen ventanas a parte sobre la principal,

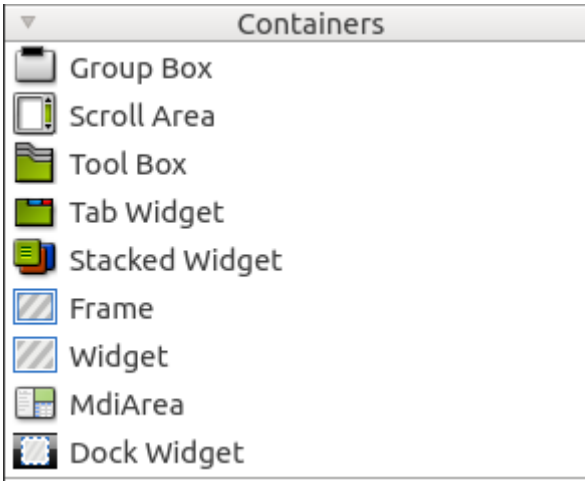


Figura 3. 10: *Containers Qt Designer*

- **Input Widgets,** son objetos cuya principal funcionalidad es la de actuar como dispositivos de entrada (figura 3.11), como la elección de un *ComboBox*, para introducir texto escrito están el *Line Edit* y el *Text Edit* que presentan ligeras diferencias visuales, por otro lado para introducir números, aparte de por teclado existen los objetos *Spin Box* y *Double Spin Box*, usados para aumentar o disminuir el valor mostrado haciendo clic sobre la flecha. Elementos para introducir fecha y hora, *Time Edit*, *Date Edit* y *Date/Time Edit*. Por último los elementos móviles como el *Dial*, las barras (*Sliders*) y las de posicionamiento (*Scrol*l).

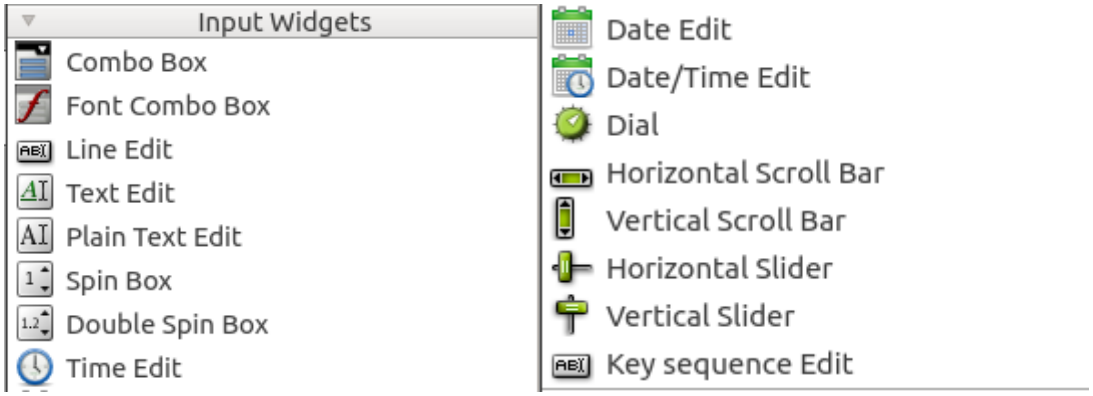


Figura 3. 11: *Input Widgets Qt Designer*

- Display Widgets (figura 3.12), usadas para mostrar datos, como el *Label*, *Text Browser*, *LCD Number*, mostrados mediante texto o de un

modo gráfico, *Progress bar*, *Calendar* y elementos meramente gráficos como las líneas horizontales y verticales.

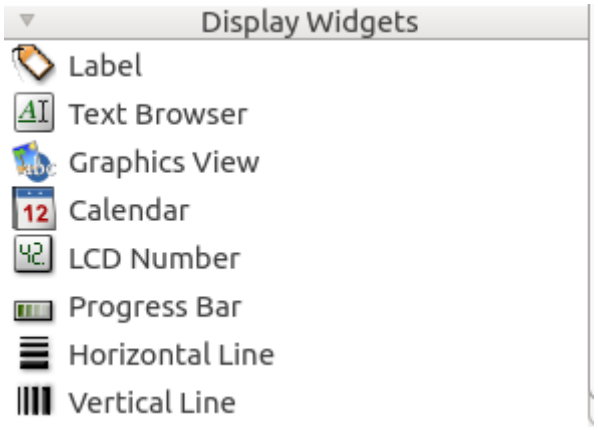


Figura 3. 12: *Display Widgets Qt Designer*

A parte de estos elementos básicos de Qt se pueden importar widgets ya creados, y que presenten una complejidad mayor.

CAPÍTULO 4. DESARROLLO DE LA INTERFAZ DE CONTROL DE iCab

El primer paso para desarrollar la interfaz de control del iCab, es realizar la configuración del entorno, para poder desarrollar una interfaz bajo la arquitectura de ROS. Primero, se instala el sistema operativo Ubuntu 14.04 en una partición del disco duro, para no eliminar el Windows 7 que contiene. Esta instalación se encuentra detallada en el anexo "Instalación del SO Ubuntu 14.04". Posteriormente se instala la plataforma de ROS y se configura el espacio de trabajo (*workspace*), que puede observarse en el anexo "Instalación y configuración inicial de ROS". Además de instalar ROS que es la comunicación del sistema, se instalará el programa de desarrollo Qt Creator, y se configurará para poder trabajar con paquetes de "catkin". Este proceso de instalación y creación del entorno de desarrollo se puede realizar siguiendo los pasos mostrados en el artículo [].

4.1. Diseño inicial de la interfaz gráfica

Antes de proceder al desarrollo de la interfaz, es necesario llevar a cabo un análisis de las necesidades funcionales del sistema para diseñar un boceto de esta, de modo que se trabaje en función de lo que necesite el control del vehículo autónomo.

La importancia de desarrollar una interfaz gráfica para controlar el vehículo iCab reside en crear la posibilidad de la intervención humana sobre este, de forma que en caso de que el vehículo presente un comportamiento anómalo pueda ser corregido por los técnicos. Además, esta interfaz no servirá simplemente para controlar el vehículo mediante órdenes, sino también para visualizar el estado en el que se encuentra mediante paneles de datos, leds, y mensajes de texto.

Las necesidades funcionales que presenta el proyecto son las siguientes:

- Conexión con el sistema de ROS, y un bloque de texto sobre el que mostrar los mensajes enviados por este (trazas).
- Botones para activar/desactivar los sistemas de control que se encuentran divididos en dos partes, tracción y dirección. Además,

tiene que ser posible elegir el modo en el que se va a controlar la tracción y la dirección, que puede ser automático o manual.

- Mostrar valores sobre la alimentación del sistema: corriente de estator y rotor, de ambos sistemas (tracción y dirección), tensión de alimentación de la placa de potencia, valores de las señales de control del sistema (PID's).

- Cuadro de control de tracción del sistema, pudiendo seleccionar el valor de la corriente que circula por el estator y por el rotor en el modo manual, y en el automático la velocidad de movimiento del sistema.

- Cuadro de control de dirección del sistema, que en el modo manual permita seleccionar el valor de la corriente (%) del rotor y del estator del motor de dirección, y en el automático, el ángulo en el que se sitúan las ruedas para hacer girar al vehículo.

- Una sección para configurar los límites de los parámetros más importantes de la etapa de potencia del sistema, que evite superarlos y provocar daños en el circuito.

- Configuración de los PID's reguladores, que se encargan de que el alcance de valores de posición y velocidad suceda con la mayor exactitud posible.

- Calibración de la sonda de efecto Hall, para el muestreo de medidas en el circuito de potencia, aplicando dos puntos para ajustar el rango de medición, obteniendo mayor o menor precisión en función de estos.

A raíz de las necesidades referidas se propuso un boceto inicial sobre el que empezar a trabajar. A continuación se muestra la figura de la pantalla inicial de la interfaz de control (Figura 4.1):

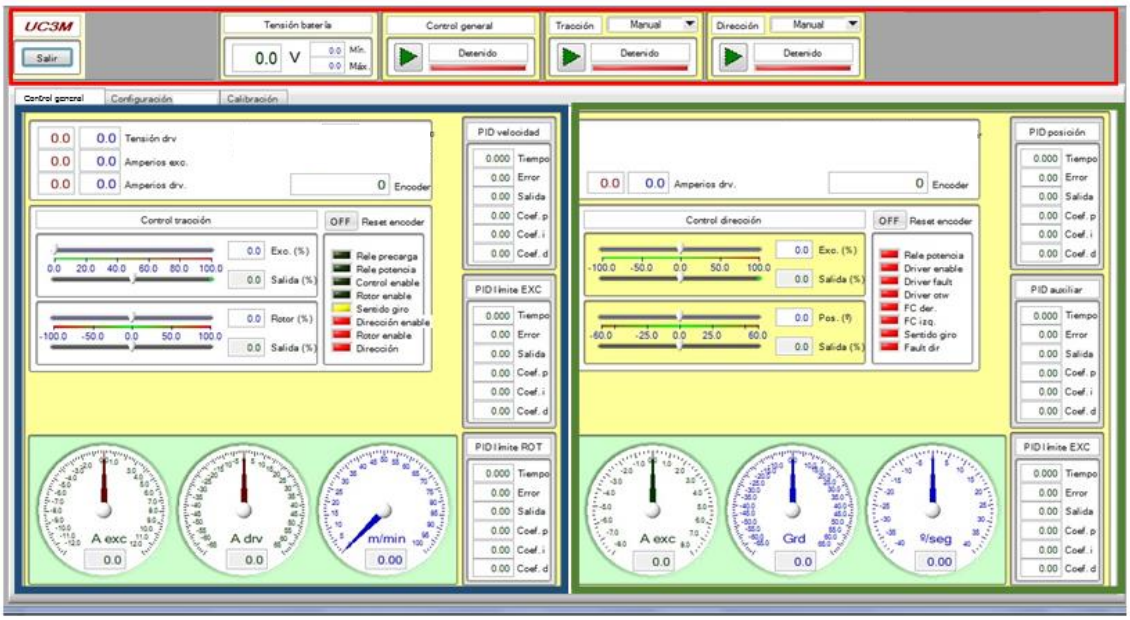


Figura 4. 1: *Primera pantalla del boceto inicial***.**

Como se puede observar en esta primera pantalla hay tres zonas remarcadas en distinto color para identificarlas. Una se corresponde al menú superior (siempre visible). Debajo de este menú hay una barra gris que presenta tres pestañas, mostrándose la primera abierta (control general) que se ha subdividido a su vez en dos áreas. Las zonas remarcadas se van a explicar a continuación:

- **Menú superior** (rojo): en este menú se disponen los botones principales del funcionamiento de la interfaz de control. Estos botones empezando por la izquierda consisten en un botón de salida que cierra la interfaz de control, un cuadro que muestra los valores de la tensión instantánea y el valor máximo y mínimo, con la que está siendo alimentado el vehículo. El siguiente botón es el que activa el control general del sistema, y los dos siguientes corresponden al control de tracción y al de dirección. Además, estos dos últimos tienen un menú de selección del modo de control del vehículo (automático o manual), que para poder activarlos es necesario que el control general se encuentre encendido.

- **Cuadro de tracción** (azul): este cuadro, situado en la primera pestaña, llamada control general, se utiliza para controlar y observar las variables del sistema de tracción. Dentro de esta, el primer

cuadro que se muestra en la zona superior sirve para visualizar los valores de alimentación del motor de tracción (tensión y corriente), y el valor actual del *encoder*. Debajo de este se sitúa el cuadro de control de tracción, dónde a través de las barras se seleccionará, en caso de estado manual, el valor deseado de estator y de rotor en porcentaje. En los cuadros de texto situados a su derecha, se indicará el valor numérico. Hay un botón "OFF" para resetear el valor del *encoder*. Bajo este se sitúan ocho leds que indican cuando se encuentran habilitados ciertos elementos de control, mientras que otros sirven para conocer el estado de los mismos. Los marcadores en forma de reloj, situados en la zona inferior, sirven para observar la corriente en amperios que circula por el estator (A exc.), la que circula por el rotor (A drv.) y la velocidad a la que se está moviendo el vehículo expresada en m/min, que son de gran importancia. Por último, a la derecha se encuentran la respuesta de los reguladores (PID's) de las señales de velocidad y de la alimentación del rotor y del estator para su observación.

- **Cuadro de dirección** (verde): este es prácticamente igual que el de tracción, pero tiene diferencias en los marcadores de aguja de la zona inferior, ya que el de en medio, sirve para observar el ángulo de giro de las ruedas del vehículo en grados y el de su derecha sirve para medir la velocidad de giro en grados/s. Además, en los recuadros de la derecha se muestran los PID's de posicionamiento, auxiliar y de alimentación del estator.

La segunda pestaña que posee la interfaz se muestra en la figura 4.2:

Figura 4. 2: *Pestaña de Configuración del boceto.*

Esta figura 4.2 se muestra la pestaña, llamada configuración, es la encargada de configurar la etapa de potencia del vehículo mediante parámetros que limitarán al sistema, con el fin de evitar el alcance de ciertos valores de corriente y tensión, que pueden provocar daños en el circuito de potencia. Se configuran los lapsos de tiempo de medida de los PID's, encargados de que se alcancen los valores indicados sin que se sobrepasen los límites. Este panel se puede dividir en cuatro partes:

- La configuración de la dirección (azul), contiene en el primer cuadro la configuración PWM de alimentación del rotor, dónde el usuario puede fijar los valores de frecuencia, valores máximos y mínimos, y la constante máxima de excitación. Debajo se muestran los parámetros de configuración del PID limitador.

- La configuración de la tracción (verde), es equivalente al de dirección, aunque, en este caso se dispone de la configuración del estator y del rotor. En la etapa de arranque la necesidad de un mayor par provoca una mayor demanda de corriente al rotor, teniendo que ser limitada para evitar superar los límites del circuito.

- La configuración general de límites (rojo), se encarga principalmente de establecer los valores máximos alcanzables, para no provocar

daños en el circuito, y los mínimos que tiene que haber para que funcione correctamente. Hay más límites en la tracción, ya que está presenta una complejidad mayor.

- La configuración de los PID's de alcance del valor requerido (naranja), regulan la posición en grados y la velocidad en cm/s. Se encargan de hacer que el alcance de estos valores se produzca con la mayor rapidez y con el menor sobre-amortiguamiento.

Por último la tercera pestaña (figura 4.3) es la encargada de calibrar el rango de medida de la instrumentación, configurando la sonda de efecto Hall, ajustando los valores correspondientes digitales con los analógicos. Para ello, mediante dos puntos se traza un intervalo de medida. Por ejemplo: con una variable digital de 256 bits (0-255), puede ajustarse para que la medida de la corriente sea de (-20, +20), siendo 0 el equivalente a -20 A y 255 el equivalente a +20 A.

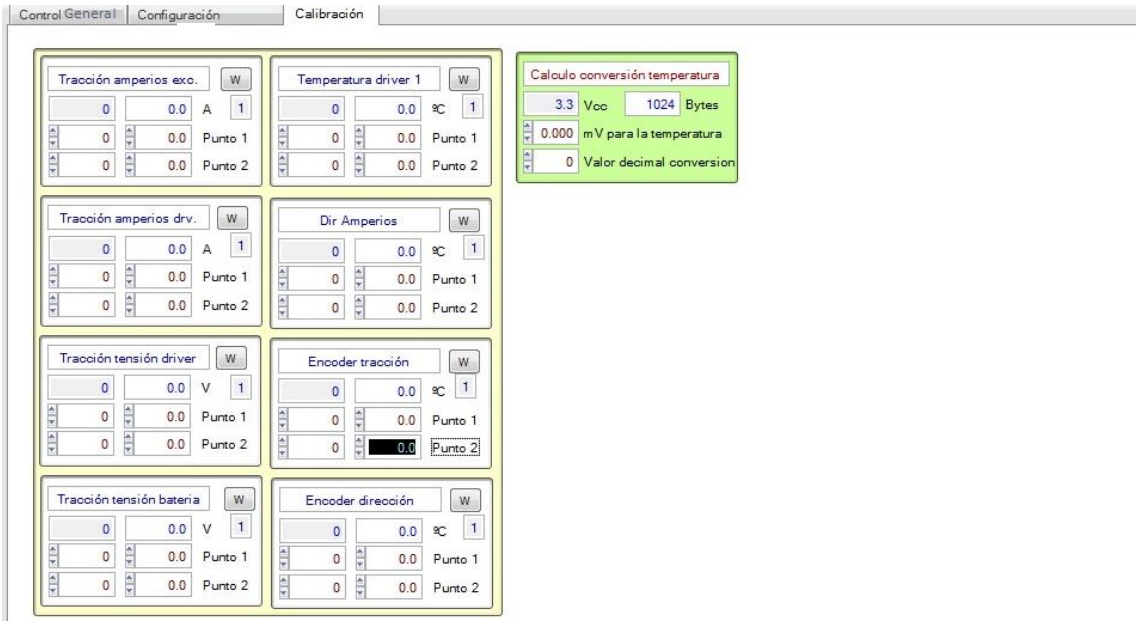


Figura 4. 3: *Pestaña de calibración del boceto.*

En la figura 4.3 se observan los apartados para distintos valores de medida, la primera columna corresponde a la tracción y la medida de tensión y corriente que circulan por el rotor y estator. La segunda presenta la temperatura que alcanza el rotor de tracción, que es el que más sufre debido a que puede alcanzar altos niveles de corriente en el arranque, en función de la situación en la que se encuentre el vehículo (pendiente de

subida). También se configura la medida de la corriente de la dirección, y los valores de ambos *encoders*, dirección y tracción. Por último, como la medida de la temperatura se efectúa midiendo el voltaje, este debe traducirse en grados, configurándose en el cuadro situado a la derecha (sombreado en verde).

4.2. Interfaz de control: icab_reconfigure

Una vez generados unos planos acordes al análisis de las necesidades funcionales para la creación de la interfaz, se precisa generar el paquete de la interfaz dentro del espacio de trabajo de "catkin". Para crear un paquete de ROS en el espacio de trabajo, hay una función llamada "catkin_create_pkg", que genera un archivo de texto llamado "CMakeList.txt" (Ver anexo IV) y otro de tipo "xml" llamado "package.xml" (Ver anexo V). El primero sirve para configurar la construcción de los paquetes, cómo construirlo y donde instalarlo [38], y el segundo contiene la información sobre el paquete como el nombre, la versión, autores, soporte técnico y dependencias sobre otros paquetes de catkin [39]. ROS dispone además de una función diseñada para su uso con Qt (C++), que se llama "catkin_create_qt_pkg", que genera los archivos anteriormente mencionados y los archivos principales para la creación de un nodo. Estas carpetas son las siguientes:

- **Include:** que incluye las cabeceras de las clases de C++, que son los archivos ".hpp" o "*headers*". Contiene dos cabeceras: main_window.hpp y <nombre del nodo>.hpp.
- **Resources:** que es donde se introducen los recursos utilizados dentro del paquete, como las imágenes.
- **SRC:** que incluye los ficheros de C++, que contienen las implementaciones de las clases principales de la aplicación. Los archivos generados son tres: main.cpp, main_window.cpp y <nombre del nodo>.cpp. Más adelante se detallará el papel desempeñado de cada fichero.

- **UI:** que incluye el archivo de la interfaz gráfica, vital para el resultado final del proyecto. Este documento ".ui" tiene un lenguaje *xml* sobre el que se desarrollan los elementos de la interfaz gracias a la herramienta de qt-designer.

Para poder cargar este paquete sobre el Qt Creator, se tiene que abrir el archivo "CMakeList.txt", que contiene la información sobre los documentos y librerías que deben añadirse al proyecto, véase anexo "CMakeList.txt".

4.2.1. Elementos visuales

Una vez generado el paquete sobre el que se trabajará, se abre el "main_window.ui" y la herramienta Qt Designer, para poder editarlo. Antes de añadir elementos a la ventana principal se añade un *Scroll Area* que permitirá que si en la pantalla no se visualiza todos los elementos, el usuario pueda desplazarse sin problema. Una vez generado se pueden añadir elementos sobre el área.

4.2.1.1. Menú superior

Se desarrolla el menú superior del trabajo con el aspecto deseado en el boceto, con ligeras modificaciones, quedando como resultado el siguiente menú (figura 4.4). Este menú es bastante intuitivo y fácil de manejar por el usuario. Además, si algo no funciona correctamente, posee restricciones lógicas que evitan operaciones indebidas.

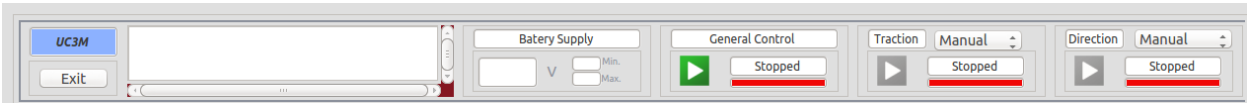


Figura 4. 4: *Menú superior de la interfaz.*

Para generar este menú se ha creado un "frame" que agrupase todos los objetos dentro de ella, formando el bloque menú superior. Este menú se compone de las siguientes partes:

- **Cuadro de salida:** contiene el nombre de la universidad y un botón "*exit*" que envía una señal cuando es pulsado a la ventana principal, provocando el cierre de la misma. Esta configuración de señales ha sido generada mediante el modo "*signals and slots*" de Qt (ver figura 4.5). El nombre de la universidad se ha desarrollado sobre un

LineEdit, en el que se ha modificado la paleta de colores para establecer el fondo azulado.

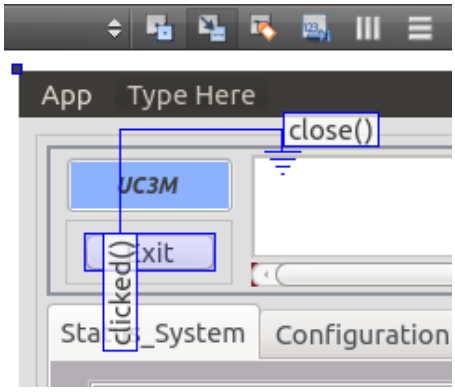


Figura 4. 5: *Signals & Slot exit button.*

- **Panel de error**: es un *ListView* generado sobre un *Scroll Area* que recibe los mensajes de error enviados por ROS, de modo que si algo no funciona correctamente se mostrarán en este cuadro.

- **Battery supply**: muestra la tensión que está suministrando la batería. Debe encontrarse entre el valor mínimo y máximo, que se muestran a su derecha para que pueda funcionar el sistema.

- **Control General**, el botón de start/stop puede ser activado solamente cuando las variables de configuración lo permitan. A la derecha del botón aparece un led que muestra el estado del botón, cambiando de color, y sobre este se muestra escrito su estado.

- **Tracción y dirección**, estos dos paneles son idénticos, pero se encargan, uno de activar la dirección y otro la tracción, y de seleccionar el modo de funcionamiento de estos. Además se muestra su estado en tiempo real, del mismo modo que en el control general.

En el desarrollo del menú superior se han utilizado los siguientes tipos de objeto:

- Para la introducción de texto se utiliza el *Label* (ver figura 4.6, letra A) para indicar alguna característica del elemento situado al lado (nombre, unidad de medida). Los *Line Edit* (ver figura 4.6, letra B), en los que se puede elegir el tipo de letra y el tamaño. Además, en la

propiedad "palette", se puede configurar los colores, como en el nombre de la universidad de fondo azul. Los elementos que muestran información tienen activada la propiedad "read only", que hace que no pueda editarse el texto por el usuario.

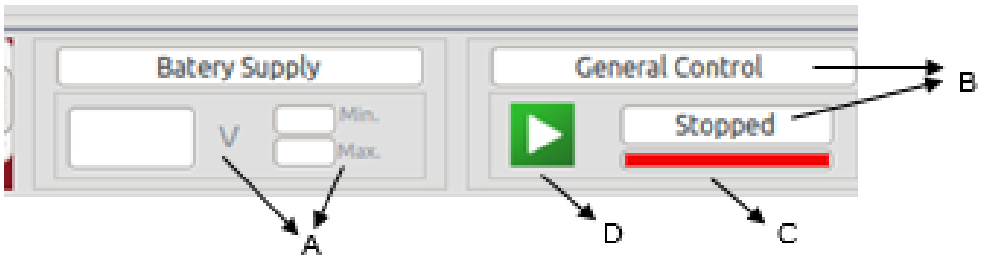


Figura 4. 6: *Diferenciación de tipos de objetos utilizados en el menú. A (Labels), B (LineEdit), C (LineEdit simulando un LED), D (PushButton).*

- Los leds que se muestran se han creado a partir de un *Line Edit* (ver figura 4.6, letra C) al que se le ha modificado la paleta de colores, simulando el funcionamiento de un led.

- La creación de los botones se ha llevado a cabo mediante el objeto *Push Button* (ver figura 4.6, letra D), modificado según las necesidades. En el caso del botón de salida, simplemente se ha introducido el nombre (Exit), en cambio, en las de "start/stop" se les ha añadido un icono en función del estado en el que se encuentra. Se modifican las propiedades del botón, mostradas en la figura 4.7, añadiendo la imagen que represente el estado del botón.

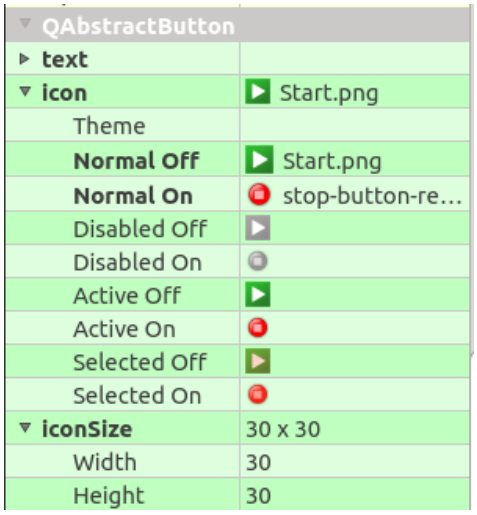


Figura 4. 7: *Properties QAbstractButton (Push Button) y los iconos identificativos.*

Después de implementar el menú superior se desarrolla el contenedor "Tab Widget", nombrado "tab_manager", y compuesto de cuatro pestañas que tienen distintas finalidades.

4.2.1.2. Status_System

Esta pestaña tiene como finalidad mostrar los datos más significativos del sistema como son las corrientes que circulan por ambos motores, la posición de las ruedas, o la velocidad a la que se mueve el vehículo. Además, es la pestaña sobre la que se puede controlar la tracción y la dirección del vehículo si estas se encuentran activadas y disponibles. El resultado final de este apartado es el siguiente (figura 4.8).

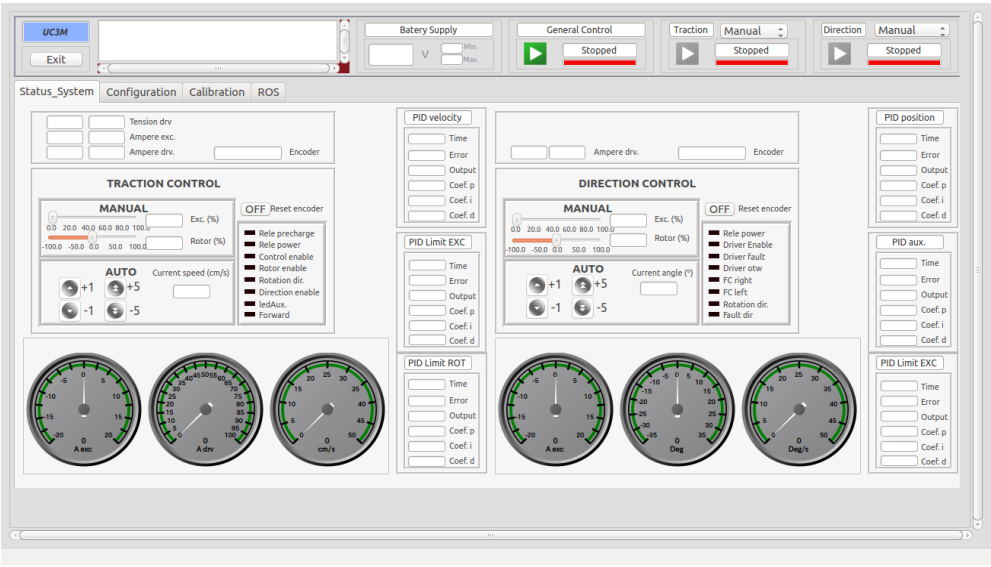


Figura 4. 8: *Primera pestaña de la interfaz "Status_System".*

Como se explicó sobre el boceto, esta pestaña se encuentra dividida en dos partes: tracción y dirección. Presenta ciertas modificaciones con respecto al boceto inicial. Dado que los cuadros de tracción y dirección son idénticos, en cuanto a los objetos que los componen, se va a explicar de un modo más detallado el de tracción para comprender el proceso de creación de ambas partes.

El primer cuadro muestra el voltaje de alimentación del rotor, las corrientes del estator y del rotor, y el valor del *encoder*. Observando la figura 4.9, compuesta por tres columnas, la primera un "LineEdit" sobre el que se indica el valor limitante de cada variable, la segunda otro *LineEdit* sobre la que se muestra el valor actual, y la tercera que es un *Label* para indicar a que se corresponde cada fila. A la derecha el *encoder* formado por un *LineEdit* para el valor y un *Label* para el nombre. En el caso del cuadro de la dirección solo presenta la corriente del rotor y el *encoder*.

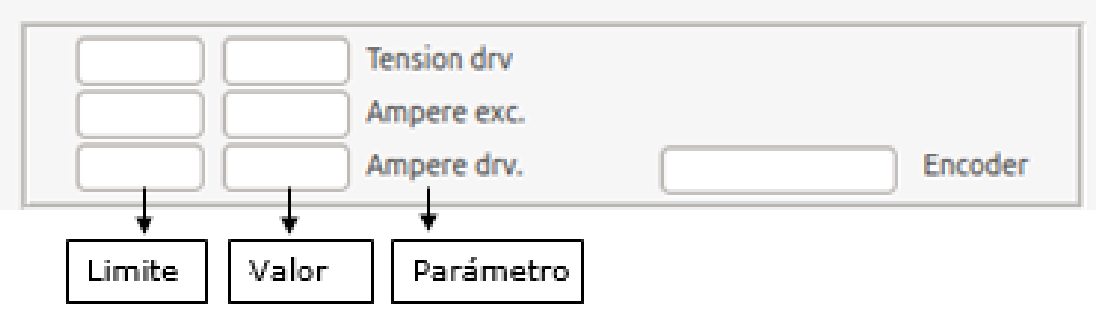


Figura 4. 9: *Identificación elementos de la interfaz.*

El cuadro de control de tracción (veáse figura 4.10), presenta un *LineEdit* para el título sin el efecto del borde, un *Push Button* llamado OFF para resetear el *encoder* con un *Label* que explica su funcionalidad, y tres cuadros que se detallan a continuación. Los cuadros que puede editar el usuario son: **manual y automático.**

El manual compuesto por dos *HorizontalSliders*, que presentan los valores porcentuales de la corriente que circula por el estator y el rotor, y a la derecha de las barras se muestra el valor porcentual de corriente medida en un *LineEdit* y un *Label* para su identificación. Este elemento se activa si se ha encendido el control de tracción en modo manual.

El automático, que funciona controlando directamente la velocidad del vehículo (cm/s), se compone de *Labels* para el título del cuadro y los valores al lado de los botones (*Push Buttons*). El incremento unitario se representa por una flecha hacia arriba, mientras que 5 incrementos se corresponden a dos flechas, las disminuciones de velocidad se representa con el icono invertido. Los botones configurados mediante iconos ayudan a la comprensión lógica, haciendo la interfaz más intuitiva.

El cuadro de leds, situado a la derecha, se ha creado del mismo modo que los leds del estado de control, descritos en el apartado anterior. Indican el estado (verde o rojo) de los parámetros de importancia listados a continuación:

- **Rele precharge**, indica que el voltaje suministrado por la batería es superior al valor mínimo (verde, superior; negro, inferior).
- **Rele power**, indica si se supera la tensión mínima de alimentación de la tracción (verde, superior; negro, inferior).
- **Control enable**, indica que se puede encender el control general porque se cumplen los requisitos. (verde, habilitado; negro, deshabilitado).
- **Rotor enable**, indica que el rotor se encuentra ya activado pudiendo ser alimentado por corriente (verde, habilitado; negro, deshabilitado).
- **Rotation dir.**, indica el sentido en la que se esta alimentando el rotor (verde, positivo; rojo, negativo; negro parado).
- **Direction enable**, indica que la dirección se encuentra habilitada y puede ser controlada (verde, habilitado; negro, deshabilitado).
- **LedAux.**, es un led que se encuentra disponible para desarrollos futuros.
- **Forward**, indica el sentido en el que se mueve el vehículo (verde, marcha adelante; rojo, marcha atrás; negro, parado).

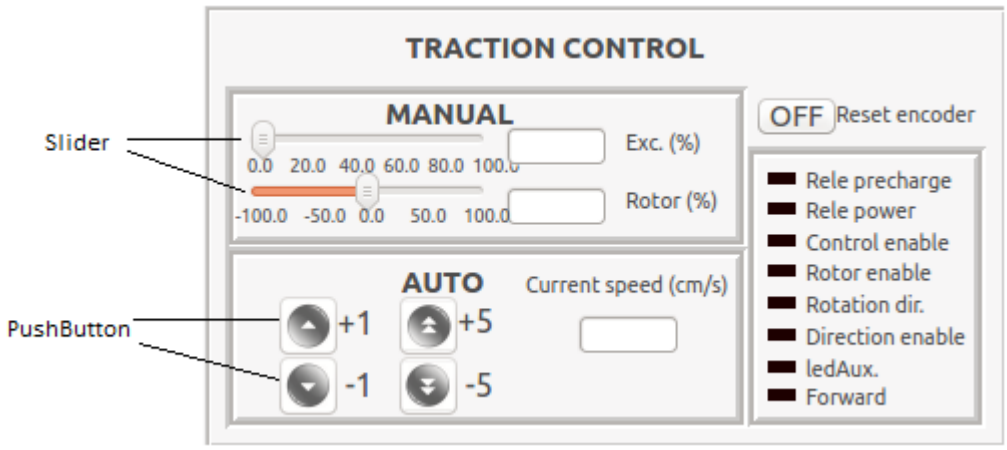


Figura 4. 10: *Panel control de tracción, modo manual y automático.*

En el cuadro de control de la dirección, el control automático regula el ángulo de giro de las ruedas que va de +35 a -35 grados, y los leds que presenta, son los siguientes:

- **Rele power**, indica cuando se supera la tensión mínima de alimentación de la dirección.
- **Driver enable**, indica cuando el rotor se encuentra habilitado (verde, habilitado; negro, deshabilitado).
- **Driver fault**, indica cuando hay un error en el rotor (verde, error; negro, sin error).
- **Driver otw**, indica cuando se produce un overflow del *encoder* (verde, error; negro, sin error).
- **FC right**, indica cuando llega a la posición límite de giro hacia la derecha (verde, límite; negro, dentro de los límites).
- **FC left**, indica cuando llega a la posición límite de giro hacia la izquierda (verde, límite; negro, dentro de los límites).
- **Rotation dir.**, indica el sentido de giro del vehículo (verde, derecha; rojo, izquierda; negro, parado).
- **Fault dir**, indica cuando se produce un error en la dirección (verde, error; negro, sin error).

A la derecha de este cuadro están los valores de los PID's configurados para alcanzar los valores deseados (figura 4.11). Se muestran los coeficientes del PID y los demás valores que son medidos en la lectura de la señal (tiempo, error y salida). En la tracción hay un PID para la velocidad, otro para el límite del estator y otro para el del rotor, y en la dirección, hay uno para la posición, otro para el estator y un último llamado auxiliar, disponible para futuras configuraciones.

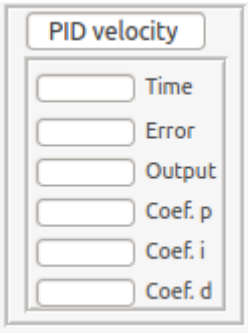


Figura 4. 11: *Lectura sobre el PID regulador de velocidad.*

Por último, el elemento más complejo que se ha utilizado en esta interfaz es el marcador de aguja (figura 4.12), generado gráficamente gracias a la importación de una clase llamada "qcgaugewidget" disponible en [40], y que consiste en una representación gráfica que sigue una estructura radial de dibujo, situando en coordenadas polares los diferentes componentes de los marcadores. En [41] se pueden encontrar ejemplos del uso de esta clase y una explicación sobre el funcionamiento de esta. Para poder desarrollar este objeto gráfico en la interfaz se ha colocado un *QVBoxLayout* por cada marcador que se necesitaba (6). Para configurar los marcadores hay que añadir a este objeto los componentes en la clase "main_window.cpp", y declarar en el header "main_window.hpp" el QcGaugeWidget, que se corresponde con el fondo del marcador y la aguja, QcNeedleItem, que se posiciona en el valor que le pasamos por parámetro. A modo de ejemplo se explica a continuación la generación del siguiente marcador.

Dentro del main_window.hpp se declaran ambos objetos, como privados:

```
private:
QcGaugeWidget * mSpeedGauge;
QcNeedleItem *mSpeedNeedle;
```

Dentro del main_window.cpp se definen ambos objetos de la siguiente forma dentro del main:

Se añade un primer fondo negro y gris, con un radio de 100.

```
mSpeedGauge = new QcGaugeWidget;
mSpeedGauge->addBackground(100);
```

Se añade un segundo círculo con un radio de 95 y negro en un extremo de la circunferencia y blanco en el otro degradándose.

```
QcBackgroundItem *bkg1 = mSpeedGauge->addBackground(95);
bkg1->clearrColors();
bkg1->addColor(0.1,Qt::black);
bkg1->addColor(1.0,Qt::white);
```

Mismo funcionamiento que el anterior, en un radio de 90 mezclando dos escalas de grises.

```
QcBackgroundItem *bkg2 = mSpeedGauge->addBackground(90);
bkg2->clearrColors();
bkg2->addColor(0.1,Qt::gray);
bkg2->addColor(1.0,Qt::darkGray);
```

Añade un arco negro en el radio 85, sobre el que se colocan las marcas de la medida.

```
mSpeedGauge->addArc(85);
```

Añade una franja de color al marcador en el radio 83.

```
mSpeedGauge->addColorBand(83);
```

Con addDegrees se ajustan las marcas de los ángulos (radio 85), con addValues se escribe el nombre de cada valor (radio 65), se añade la etiqueta del marcador (radio70), y se añade por último el *Label* sobre el que se muestra el valor del marcador.

```
mSpeedGauge->addDegrees(85)->setValueRange(-40,40);
mSpeedGauge->addValues(65)->setValueRange(-20,20);
mSpeedGauge->addLabel(70)->setText("A exc");
QcLabelItem *lab = mSpeedGauge->addLabel(55);  lab->setText("0");
```

Se añade la aguja al marcador y se especifica la longitud (70), que se ajuste al valor que marque el *Label*, el rango de medición para ajustar el

ángulo a los valores, y por último el eje central se añade un círculo gris oscuro.

```
mSpeedNeedle = mSpeedGauge->addNeedle(70);
mSpeedNeedle->setLabel(lab);
mSpeedNeedle->setColor(Qt::white);
mSpeedNeedle->setValueRange(-20,20);
mSpeedGauge->addBackground(10);
```

Se incorpora a la interfaz el elemento creado.

```
ui.A_exc->addWidget(mSpeedGauge);
```

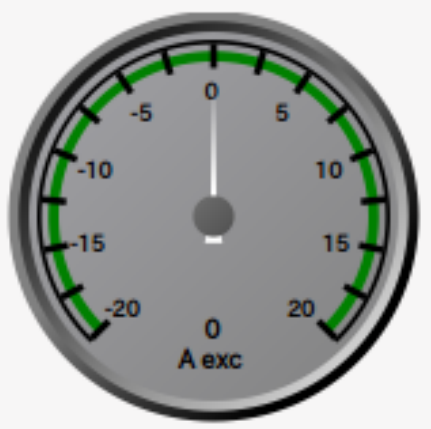


Figura 4. 12: *Marcador de aguja qcgaugewidget.*

Dentro de la clase qcgaugetwidget se pueden modificar los elementos que se han añadido a este objeto. El color de la banda, puede crearse a varios colores, indicando una franja de peligro o límite, sin embargo, esta debe ser común a todas las que se implementan, por lo que no se ha podido aplicar este uso porque las franjas de riesgo en algunos marcadores se situaban al final y en otros casos, como el del ejemplo, es necesario indicarla tanto al principio del arco, como al final.

Esta pestaña en ejecución muestra los valores como en la figura 4.13 que se muestra a continuación.

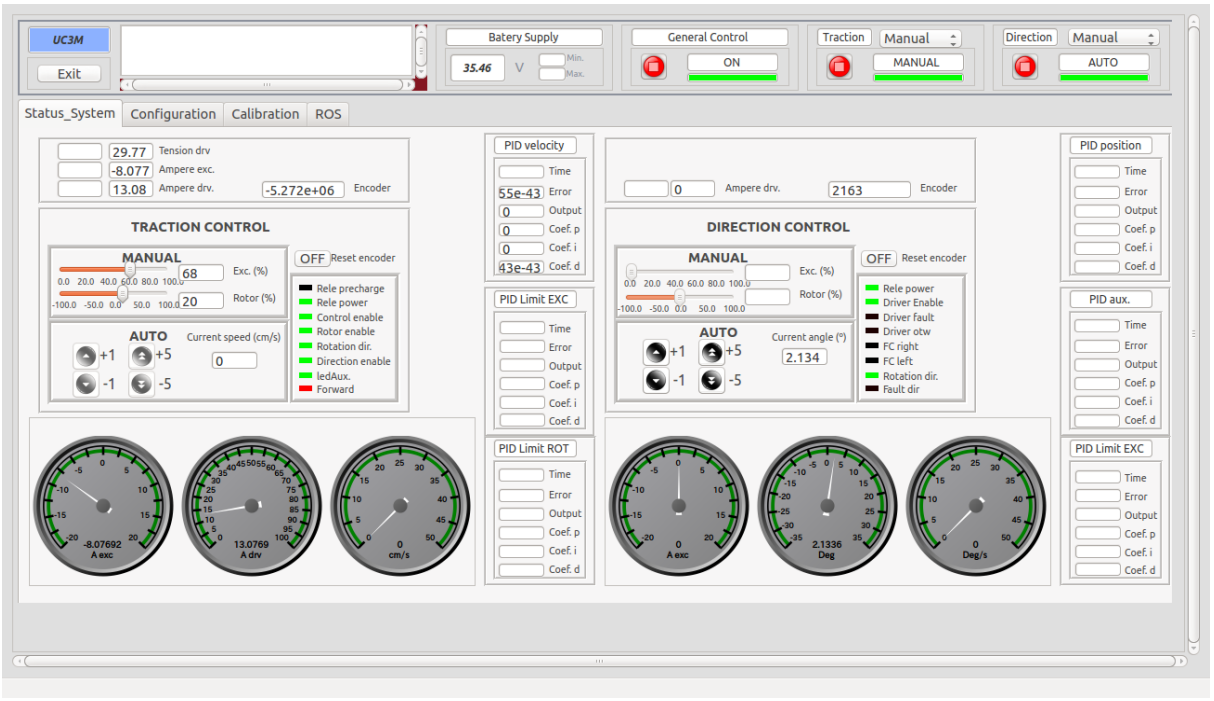


Figura 4. 13: *Status_System funcionando en una simulación con un archivo ".bag" en ejecución, encargado de generar las señales de ROS.*

4.2.1.3. Configuration

Es la segunda pestaña del "tab_manager" (figura 4.14), encargada de configurar los valores limitantes de PID's y alimentación del sistema. El cuadro finalmente presenta el siguiente aspecto.

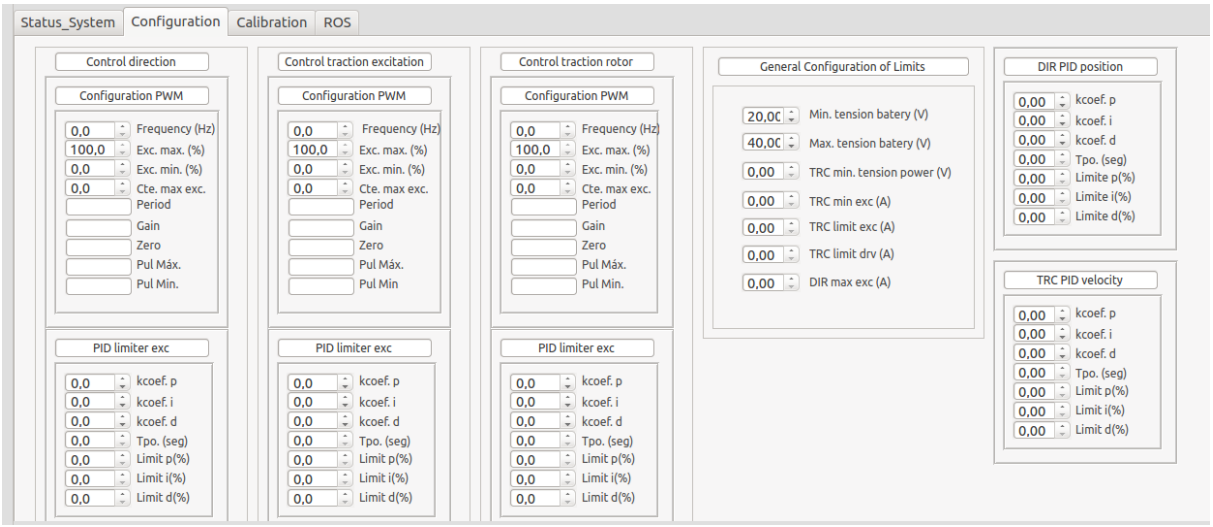


Figura 4. 14: *Pestaña Configuration de la interfaz gráfica.*

Como se explicó en el boceto, esta pestaña posee distintos cuadros para configurar los PID's que regulan las señales del sistema. Además, se

configuran los límites de las señales PWM de la corriente suministrada a cada elemento del motor (rotor y estator). Por último, posee la configuración general de alimentación de la placa, con el voltaje mínimo y máximo, potencia mínima para traccionar, corrientes máximas (límitantes) y mínimas de tracción, y el límite de la corriente de dirección.

Principalmente compuesta por "DoubleSpinBoxes", que sirven para introducir valores por teclado y aumentarlos o disminuirlos mediante las flechas que posee. Estos objetos tienen que ser configurados (figura 4.15), determinando los valores máximos y mínimos alcanzables y el intervalo con el que se cambia el número.

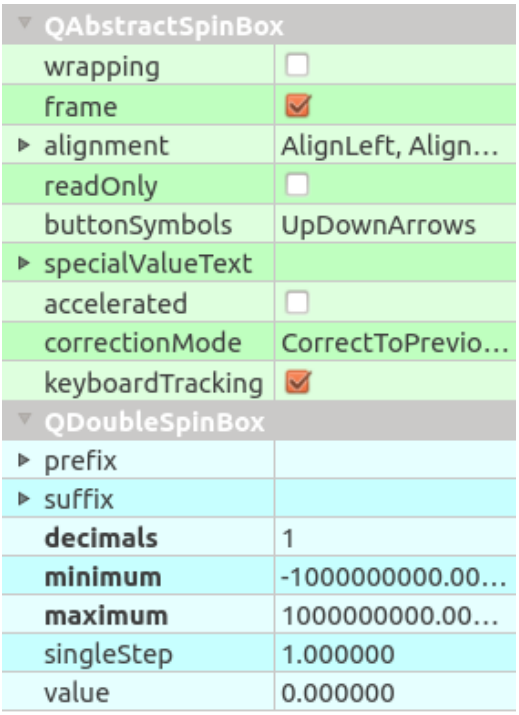


Figura 4. 15: *Properties SpinBox.*

4.2.1.4. *Calibration*

Es la tercera pestaña (figura 4.16), encargada de configurar los intervalos de medida de la sonda de efecto Hall, para conseguir una mayor precisión en el rango de interés. El cuadro implementado se muestra a continuación.

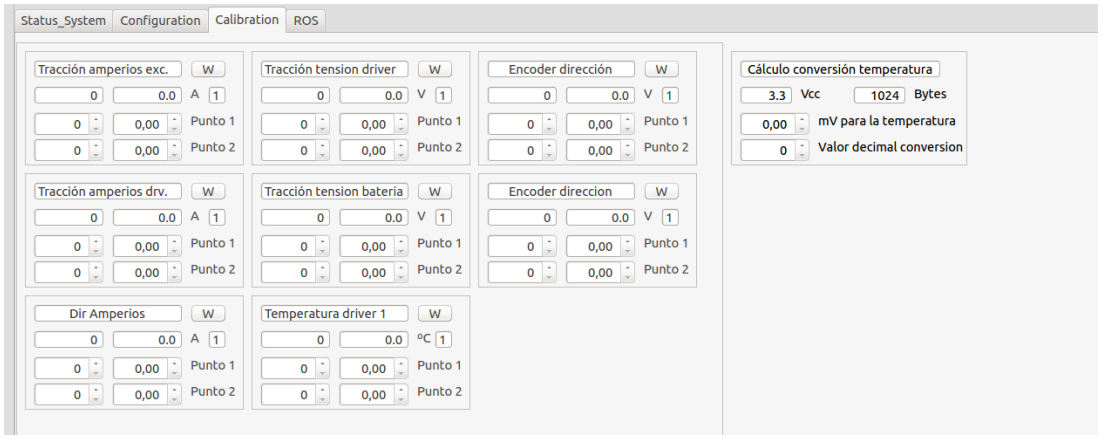


Figura 4. 16: *Pestaña Calibration del "tab_manager".*

Los cuadros de calibración tienen el mismo aspecto por lo que se explica a continuación la creación del primero (figura 4.17), a modo de ejemplo.

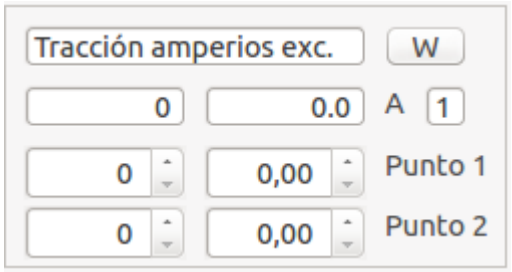


Figura 4. 17: *Cuadro de ajuste de la calibración para medir la corriente de excitación.*

La primera fila está compuesta del título del cuadro creado con un *LineEdit* y un botón *Push Button* para el envío de los cambios. La segunda fila se compone de dos *LineEdit* que muestran el valor actual en el conversor analógico digital, siendo el primero el valor digital y el segundo el analógico, y al lado se muestra la unidad de medida con un *Label*. La tercera y cuarta línea es la implementación de los valores de los puntos equivalentes, creados con *SpinBoxes*, que componen la recta de calibración.

Por otro lado se dispone del calibrado sobre la conversión de temperatura, medida en tensión.

4.2.1.5. ROS

Esta pestaña contiene la conexión a las señales de ROS (figura 4.18) creada por el paquete original del "catkin_create_qt_pkg", y transportada a esta pestaña. Consiste en una serie de líneas que sirven para efectuar si fuese

necesario una conexión tcp/ip. En el caso de este proyecto simplemente hay que pulsar el botón "Connect" que presenta.

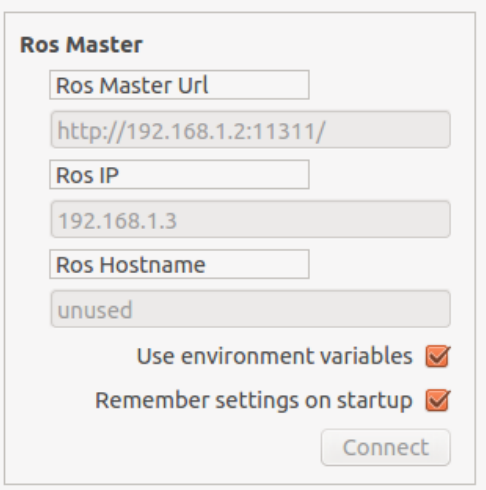


Figura 4. 18: *Panel de conexión a ROS, situado en la última pestaña.*

Hasta este momento la interfaz generada tiene una distribución amigable e intuitiva para el usuario, sin embargo, aún es necesario "darle vida" a todos los objetos de la interfaz gráfica, introduciendo relaciones lógicas que doten al sistema de cierta inteligencia. Mediante la activación/desactivación de botones, invocando servicios mediante el envío de peticiones al sistema a través del nodo "icab_reconfigure", y mostrando todos los mensajes enviados desde el nodo "movement_manager".

4.2.2. Lógica de la interfaz y conexión de señales

Las interfaces deben estar dotadas de cierta inteligencia para poder anticiparse al usuario, como indicaba **Licklider** (1960), además debe mostrar en tiempo real el estado de todos los elementos del sistema, así como recibir mensajes de error para mostrar por pantalla.

Las clases de las que se compone el paquete del "icab_reconfigure" tienen distintas funcionalidades, ya que el archivo qnode.cpp es el encargado de configurar todo lo relacionado con ROS (servicios, subscripciones). Por otro lado está el main_window.cpp, que es el archivo, sobre el que se genera la lógica de la interfaz, actuando directamente sobre los objetos que posee.

Para que el usuario interactúe con el ordenador es necesario configurar el desarrollo del envío de señales, en la mayoría de los casos, esto se produce

cada vez que un botón es pulsado. A continuación, se empezará por la explicación sobre las señales que son recibidas por la interfaz y posteriormente las señales que se envían a través de la interfaz.

4.2.2.1. Señales de entrada

El sistema recibe señales provenientes del topic "Status_Info", publicado por el nodo "movement_manager", al que se encuentra subscrito el nodo de la interfaz, llamado "icab_reconfigure. Esta comunicación efectuada por ROS se presenta en la siguiente figura del rqt_graph.

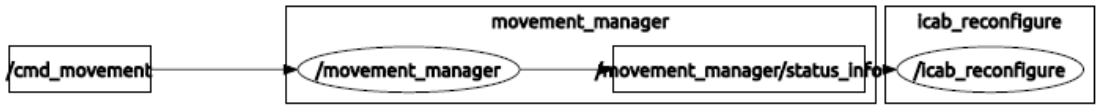


Figura 4. 19: *Gráfico sobre la conexión de los nodos, topics y mensajes de ROS mediante ROS Graph.*

Como se ha indicado anteriormente, el topic "/movement_manager/status_info" envía una estructura de datos, generada en el "movement_manager.cpp" con la función "publishState" que contiene el estado en el que se encuentra el vehículo envíado por la placa de control. Los datos enviados por el "topic" tienen que ser recogidos por el nodo a través de una clase llamada qnode, que es utilizada por la clase de la ventana principal del sistema (main_window) para traspasar estos valores a los elementos visuales del sistema. En función del valor obtenido, el sistema efectuará una serie de acciones en consecuencia.

La clase qnode tiene una funcion llamada "updateCb" al la que se le pasa por parámetro el topic "StatusInfo" del "movement_manager", la clase qnode posee una estructura privada llamada "TGStatus", con la que se ha creado la estructura "global_status_" que recoge los valores del parámetro.

Esta clase sirve de punto intermediario entre la ventana principal y el "status_info". Para poder traspasar los valores a los elementos de la interfaz en el "main_window.cpp" se realiza la llamada a la función creada "updateStatus", que pasa por parámetro la estructura general_status que recibe los valores de la clase qnode donde esta declarada la estructura.

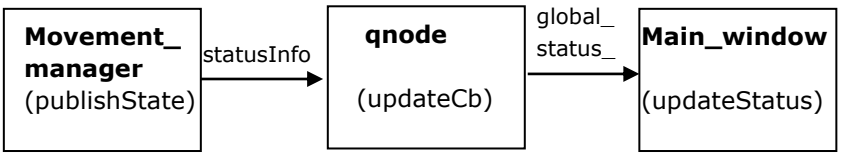


Figura 4. 13: *Traspaso del estado del sistema por los distintos módulos.*

A lo largo de esta función se identifican los elementos visuales con su correspondiente valor y además se efectúan operaciones lógicas en función de los valores recibidos. Lo primero que se va a explicar es los estados del control del vehículo y su funcionamiento.

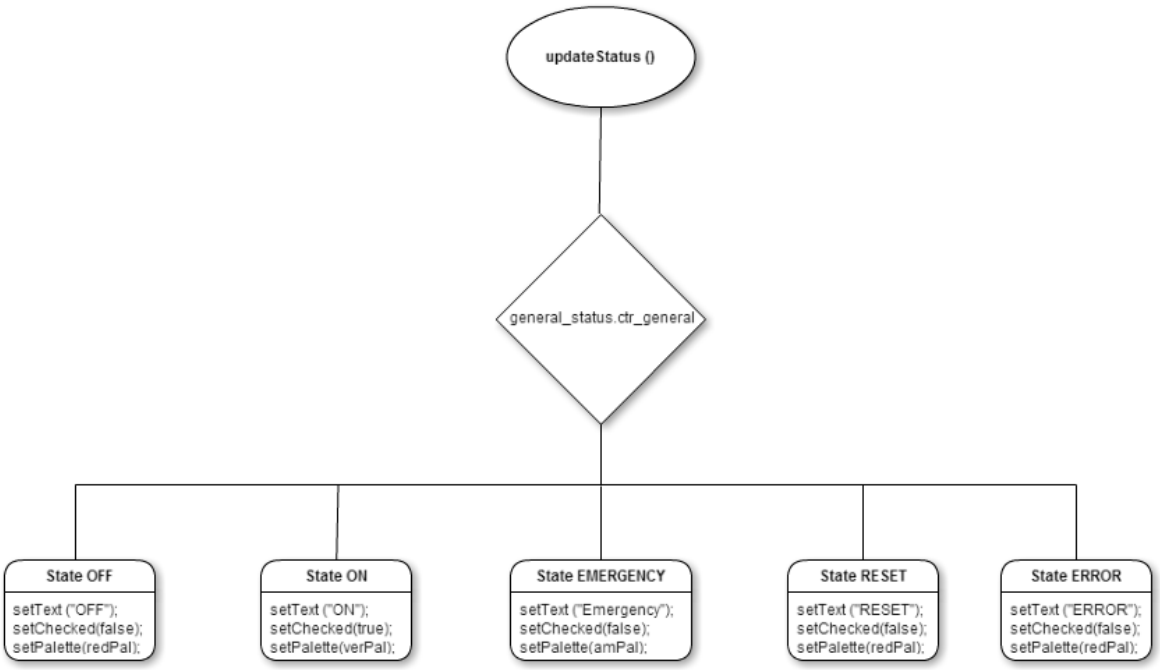


Figura 4. 14: *Diagrama de flujo sobre el estado del control general*

Como se puede observar en el flujograma se realiza un switch sobre la variable de control general, que indica el estado en el que se encuentra, actuando en consecuencia en función del estado:

- **OFF**, el control general se encuentra apagado, y por consiguiente se deshabilitan el control de tracción y el de dirección, de forma que no

pueda enviarse peticiones de activación del sistema. Además se pone el led en rojo y se indica el estado en el que se encuentra "Stopped".

- **ON**, el control general se encuentra encendido, y por consiguiente se activan los botones que activan/desactivan el control de traccion y el de dirección. Además se pone el led en verde y se indica el estado en el que se encuentra "ON".

- **EMERGENCIA**, ha sucedido un imprevisto y ha saltado el sistema de seguridad del iCab (se ha pulsado la "seta" de emergencia), por lo que requiere que se reactive el sistema, se desactivan todos los botones de control. Además se pone el led en ámbar e indica por pantala el estado de emergencia.

- **RESET**, despues del estado de emergencia el control general debe pasar a este estado, necesitando que se haga un reset sobre este antes de volver al estado inicial OFF. El led se enciende en rojo y se muestra por pantalla el estado.

- **ERROR**, este caso sucede por algun problema en la comunicación, ya que no se ha recibido ningún mensaje coherente con los casos anteriores. Se establece por defecto, y desactiva los controles para que no se pueda seguir manipulando el sistema. Se deberá proceder a la revisión del error, y se indica por pantalla el mensaje y el color del led en rojo.

El siguiente que se va a explicar es el del control de tracción, que presenta una complejidad mayor, ya que posee tambien el modo de funcionamiento. Para analizar su comportamiento se parte de que el control general se encuentra encencido, así el boton se encontrara habilitado.

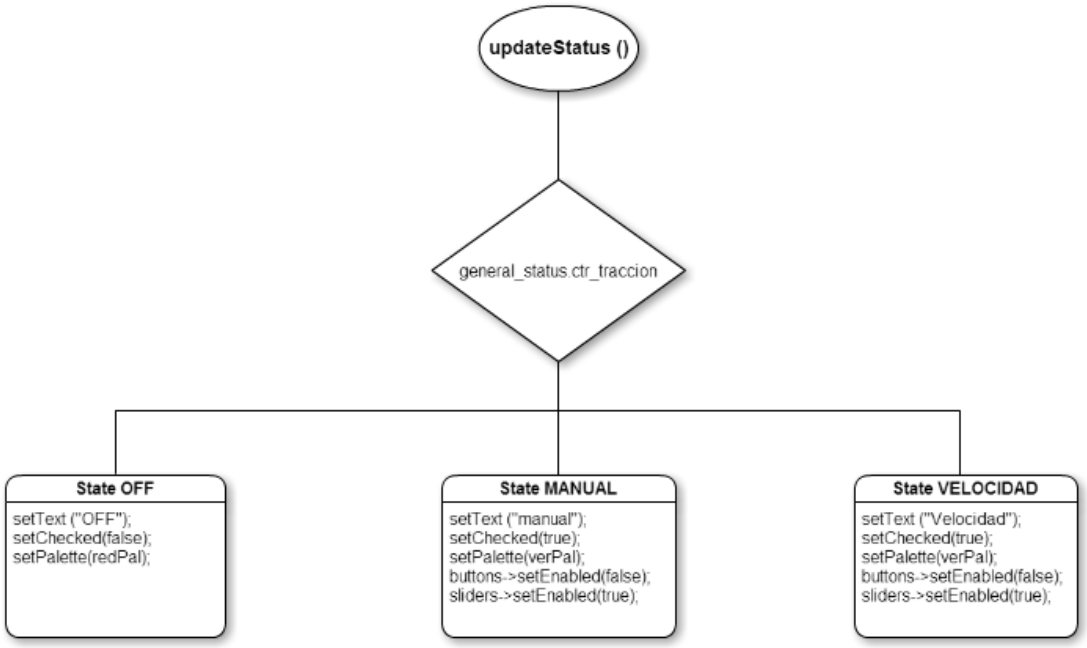


Figura 4. 22: *Diagrama de flujo sobre el estado del control de tracción*

Como se observa en el flujograma la variable del control de tracción es usada en un switch para mostrar el estado en el que se encuentra el sistema. Las opciones observadas son:

- **OFF**, el control de tracción se encuentra apagado, y por consiguiente se deshabilitan el control de tracción y el de dirección, de forma que no pueda enviarse peticiones de activación del sistema. Además se pone el led en rojo y se indica el estado en el que se encuentra, "OFF".

- **MANUAL**, el control de tracción se encuentra encendido, y el combobox está en modo manual, se desactivan los botones del modo automático y se activan los "sliders" del manual. Además se pone el led en verde y se indica el estado en el que se encuentra, "MANUAL".

- **VELOCIDAD**, el control de tracción se encuentra encendido, y el combobox está en modo auto, se activan los botones del modo automático y se desactivan los "sliders" del manual. Además se pone el led en verde y se indica el estado en el que se encuentra, "VELOCIDAD".

El de dirección tiene un comportamiento similar al de tracción, salvo con la diferencia de que en vez de modo de VELOCIDAD, tiene el modo de POSICIÓN, en el que se activan los botones correspondientes al manejo automático del sistema de dirección, así como se deshabilitan y habilitan los "sliders", que controlan el porcentaje de corriente que circula por el rotor y estator. En la posición, a la hora de activar o desactivar botones, se ha implementado una comparación sobre el ángulo actual que bloquea los botones una vez que se ha alcanzado el valor máximo o mínimo de posición. Por consiguiente, por ejemplo, en el caso en el que la posición de la dirección esté en 31º (límite 35º), se deshabilitara el botón que envía la petición de sumar 5 grados a la posición actual.

En relación a los leds, estos son iluminados en verde o rojo en función del valor de las variables booleanas que las representan, mediante una función if-else se cambiara el color de led a través de la función setPalette().

Por último se rellenan las casillas correspondientes con los valores enviados en el global_status_, para lo que se edita el texto de los casilleros, como por ejemplo en el voltaje de la batería.

```
ui.BatteryLineEdit->setText(QString::number(general_status.val_ten_battery, 'g', 4));
```

Con todas estas señales de entrada se hace llegar al usuario toda la información correspondiente sobre el estado del vehículo, permitiendo que se pueda actuar en consecuencia si algo no funciona correctamente. Para actuar sobre el vehículo es necesario formar un medio de envío de señales sobre el que enviar peticiones al sistema, y además, configurar los servicios, que son llamados a través de los botones de interactuación del usuario con el sistema.

4.2.2.2. Señales de salida

Para configurar las señales de salida es necesario disponer de un medio sobre el que enviarlas al vehículo, llamado "service client call", creado por el nodo icab_reconfigure (qnode), y que será publicado en el topic "cmd_movement" (véase figura 4.19), al que se subscribe el movement_manager, generando una serie de tareas a realizar por el vehículo.

A continuación se van a explicar una serie de señales de salida de los elementos más interesantes. Empezando por el botón de arranque del control general.

Para poder activar el control general del vehículo la primera condición que se aplica es que la conexión de ROS se encuentre activada, y una vez que esto se ha comprobado puede haber dos casuísticas sobre el control general:

- STOP: se para el control general si su estado es distinto al de OFF. En este momento se deshabilitan los controles de tracción y dirección, y se efectúa la llamada al servicio de qnode llamado statusGeneralCall(CONTROL_GRAL_OFF). A esta función se le pasa por parámetro el estado al que se desea hacer pasar al control general, que han sido definidos previamente.

- START: si su estado es el de OFF, para mayor seguridad se comprueba que no corresponde con el estado ON, y se activan los botones del control general y el de tracción, y se genera la llamada al mismo servicio que en el STOP, pero en este caso se envía por parámetro el valor del "CONTROL_GRAL_ON".

Otro funcionamiento interesante es la generación de señales al producir ciertas acciones, como las que se producen al mover los sliders del control manual de la tracción cuando estos se encuentran activados. Primero se lanza una señal cuando se mueva el slider pasando por parámetro el valor que tiene, en la función generada por la señal se realiza la llamada al servicio setTraction, que posee tres parámetros de entrada, el modo de funcionamiento (ComboBox), el valor del estator (slider_value), y el del rotor (slider_value). El servicio se encarga de generar una petición al movement_manager, que si es devuelta satisfactoriamente significará que el valor se ha modificado correctamente.

En el caso del control de dirección automático, se puede modificar la posición del ángulo mediante los botones de subida y bajada, que encontrándose a priori activos, el sistema reaccionara una vez que se pulsen, generando una señal (on_turnRightOneButton_clicked()), que se

traduce en una función desde la que se llama al servicio setDirection(int mode, float value), en la que pasando el modo CONTROL_DIR_POSICION y el ángulo actual más los ángulos que se suma o resta a la posición original (angle+1), el servicio envía la petición con estos datos, y si se genera una respuesta positiva, se observa si el valor del ángulo de respuesta es igual que el ángulo de la petición. Si algo falla se enviará un error a través de ROS.

Las señales que se han creado para cada botón están definidas por su nombre, siguiendo la estructura on_<name_object>Button_clicked (), también pueden generarse a través del uso de la herramienta "Go_to_Slot()", genera la función llamada por la señal producida.

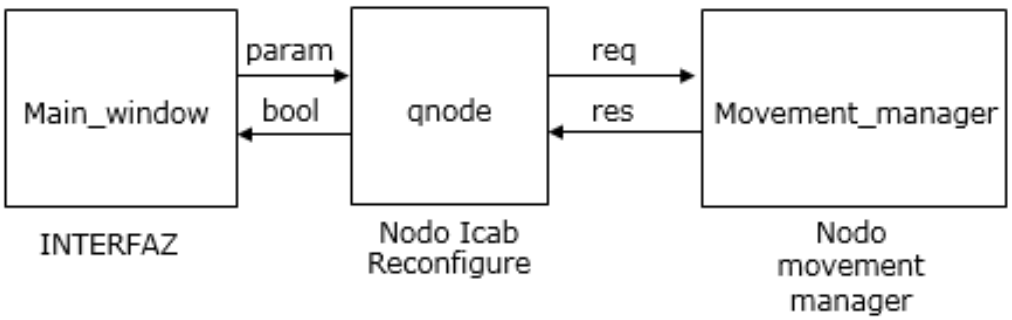


Figura 4. 23: *Gráfico sobre el envío de señales a través de los módulos del sistema.*

Como se ha podido observar el funcionamiento de los mensajes de salida del sistema siguen un procedimiento general, que se puede dividir en los siguientes pasos:

1. A través de un objeto de la interfaz se genera una señal, cuyo punto final es una función implementada en el "main_window.cpp".

2. Desde la función se efectúa la llamada a un servicio de la clase qnode pasando por parámetro las variables que componen el mensaje de petición.

3. El servicio de la clase qnode genera un mensaje de petición con los valores introducidos, y espera una respuesta para comprobar si se ha producido correctamente el servicio.

Tras configurar las señales de salida el usuario ya puede interactuar con el vehículo. La interfaz ya está dotada de las capacidades y limitaciones necesarias para que pueda actuar sobre el iCab. Es posible añadir elementos e instrucciones a la interfaz, si el diseño más adelante lo requiere.

CAPÍTULO 5. CONCLUSIONES Y TRABAJOS FUTUROS

La interfaz desarrollada para el control del vehículo autónomo iCab ha alcanzado una serie de objetivos indispensables para el buen funcionamiento del sistema.

La interfaz es capaz de conectarse a la comunicación de ROS, mediante la que recibe los datos provenientes de la placa de control del vehículo. Estos datos son mostrados de una forma ordenada, clara y con símbolos intuitivos, permitiendo al usuario observar rápidamente el estado del vehículo autónomo. Se puede configurar la resolución con la que se miden los parámetros del sistema, ajustando el rango de medición.

Por otro lado la interfaz permite el control general del sistema siempre que reciba las condiciones necesarias para activarlo, e impide su control en el caso contrario. Una vez que se activa el control general, se permite al usuario activar la tracción y dirección del sistema, y en función del modo en que la deseen activar podrá efectuar modificaciones sobre unos parámetros u otros. El control automático de ambos sistemas permite ajustar la posición del ángulo de las ruedas y la velocidad con la que se mueve el vehículo.

A parte de enviar órdenes al vehículo, se puede configurar en la etapa de control los reguladores de señal para que el envío se produzca de una forma eficaz y segura, ya que se configuran también los puntos limitantes de los parámetros críticos del sistema, como es la corriente que demanda el circuito de tracción (más elevada que la de dirección).

A través de la interfaz gráfica se puede explorar las distintas capacidades del vehículo autónomo iCab, pudiendo analizar su comportamiento ante las diferentes configuraciones introducidas.

En resumen, la interfaz gráfica desarrollada es un sistema visual intuitivo y amigable, que facilita al usuario el control del vehículo y permite el rápido aprendizaje sobre el uso de este.

Futuros desarrollos

En futuras evoluciones sobre la interfaz se podrán añadir una serie de aspectos interesantes sobre el vehículo:

- Representación gráfica sobre las señales del sistema, mediante la herramienta de ROS rqt_graph, pudiendo observar el funcionamiento interno de la arquitectura ROS.
- Incorporación de la imagen real del vehículo a través de la cámara frontal del iCab, ya incorporada, permitiendo al usuario observar la dirección que sigue el vehículo sin necesidad de estar presente en su ubicación.
- Incorporación de la lectura de sensores de distancia, para poder avisar al usuario sobre la proximidad de objetos, permitiéndole configurar el margen de seguridad establecido.
- Creación de mapas generados por un sistema GPS, pudiendo configurar puntos de destino y rutas de acceso. Para esta aplicación el vehículo debe disponer de un sistema de localización de alta precisión y ser capaz de desarrollar rutas alternativas.

CAPÍTULO 6. PRESUPUESTO

En el presupuesto que se presenta a continuación se muestra los elementos que se han utilizado para la realización de este proyecto.

Ítem	Descripción técnica de equipos y materiales requeridos	Q	Precio (material + instalación)	Precio total Equipos o Materiales
1	Hp Pavilion Dv6-3183es (2011). 4 GB de memoria DDR3 (1 x 4.096 MB). unidad de disco duro SATA de 500 GB (7.200 rpm), procesador Intel Core i7-720QM a 1,6 GHz y tarjeta gráfica ATI Mobility Radeon HD 5650[42]	1	100 €	100 €
2	SO Ubuntu 14.04	1	0	0
3	ROS-Indigo Desktop-full	1	0	0
4	Qt Creator 5	1	0	0
5	Horas de desarrollo	100	6,25 €	725 €
TOTAL				750 €

Tabla 6.1: *Presupuesto del proyecto de la Interfaz gráfica.*

La mayoría de los recursos son de acceso libre, por lo que los únicos costes son: el portátil sobre el que se efectúa el desarrollo de la interfaz y la mano de obra. El portátil tiene un ciclo de vida de 3 años de media, por lo que un portátil de estas características que cuesta 600€ y su uso ha sido durante seis meses (1/6 de la vida media) supone un coste final de 100€. El coste unitario por hora de desarrollo, ha sido estipulado sobre el sueldo de un becario (6,25€/hora).

Este coste podría ser menor si se utilizase un portátil de menor categoría, ya que es para el uso de ROS y Qt sobre un sistema operativo Ubuntu, que no requiere un portátil de más de dos Gb de Ram, ni un procesador de alto nivel como el i7. Además, este proyecto se realizó sobre una partición del disco duro de 50 Gb, gracias a que el SO Ubuntu es muy ligero (menor de 10 Gb).

CAPÍTULO 7. BIBLIOGRAFÍA

[1] Página web de ROS [Online] [Consulta: 20 de mayo de 2015] http://www.ros.org/history/

[2] *Diccionario de la lengua española*, 23ª edición [Online] [Consulta: 20 de mayo de 2015] http://www.rae.es/

[3] V. Bush. (1945, julio). *As we may think*. [Online] [Consulta: 20 de mayo de 2015] http://web.mit.edu/STS.035/www/PDFs/think.pdf

[4] J. C. R. Licklider. (1960). *Man-Computer Symbiosis*. [Online] [Consulta: 20 de mayo de 2015] http://worrydream.com/refs/Licklider%20-%20Man-Computer%20Symbiosis.pdf

[5] J. C. R. Licklider y W. Clark. (1962). *On-Line Man-Computer Communication.* [Online] [Consulta: 20 de mayo de 2015] http://cis.msjc.edu/courses/internet_authoring/CSIS103/resources/ON-LINE&20MAN-COMPUTER%20COMMUNICATION.pdf

[6] D. C. Engelbart. (1962). *Augmenting Human Intellect: A Conceptual Framework* . [Online] [Consulta: 30 de mayo de 2015] www.1962paper.org/web.html

[7] W. J. Hansen. "User Engineering Principles for Interactive Systems" *en Fall Joint Computer Conference*, NY, 1971, pp. 523-532.

[8] M. A. Hiltzik, "Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age" Ed. New York: HarperCollins, 1999.

[9] T.A. Wadlow. (1981). *The xerox alto computer*. [Online] [Consulta: 20 de mayo de 2015] www.guidebookgallery.org/articles/ thexeroxaltocomputer

[10] J. A. Incera. (2007). *Nuevas Interfaces y sus Aplicaciones en las Tecnologías de Información y Comunicaciones* [Online] [Consulta: 30 de mayo de 2015] http://www.researchgate.net/publication/240618410_ Nuevas_Interfaces_y_sus_Aplicaciones_en_las_Tecnologas_de_Informa cin_y_Comunicaciones.

[11] Mac History.net [Online] [Consulta: 15 de junio de 2015] http://www.mac-history.net/apple-history-2/apple-lisa/2007-10-12/apple-lisa

[12] C. Marrero Expósito. (2006*). Interfaz gráfica de usuario: Aproximación semiótica y cognitiva*. [Online] [Consulta: 30 de mayo de 2015] www.chr5.com/investigacion/investiga/igu/igu_aproximacion_semio-cognitiva_by_chr5.pdf

[13] Historia de Windows. [Online] [Consulta: 15 junio 2015] http://windows.microsoft.com/es-es/windows/history

[14] D. Ryan. (2011). *History of Computer Graphics: DLR Associates Series.* [Online] [Consulta: 20 de mayo de 2015] http://file181.luteebook.org/mi9lo_history-of-computer-graphics-dlr-associates-series.pdf

[15] Operating System Interface Design Between. (1981-2009). [Online] [Consulta: 30 de mayo de 2015] http://www.webdesignerdepot.com/2009/03/operating-system-interface-design-between-1981-2009/

[16] D. Mark. (1997). *Getting to know NEXTSTEP Mac Tech* Getting to know [Online] [Consulta: 15 de junio de 2015] http://www.mactech.com/articles/mactech/Vol.13/13.07/Jul97-GettingStarted/index.html

[17] AMIGA WORKBENCH 2.04 [Online] [Consulta: 20 de mayo de 2015] http://www.webdesignerdepot.com/2009/03/operating-system-interface-design-between-1981-2009/

[18] Proyecto GNOME. [Online] [Consulta: 30 de mayo de 2015] http://www.gnome.org/

[19] AQUA [Online] [Consulta: 30 de mayo de 2015] http://apple.wikia.com/wiki/Aqua

[20] Mac OS Leopard. [Online] [Consulta: 30 de mayo de 2015] http://www.applesfera.com/apple/mac-os-x-leopard-interfaz

[21] Todo sobre OSX Yosemite Applesfera. [Online] [Consulta: 20 de mayo de 2015] www.applesfera.com/tag/todo-sobre-os-x-yosemite

[22] Historia de Ubuntu. [Online] [Consulta: 20 de mayo de 2015] https://gabrielvegas.wordpress.com/ubuntu-history/

[23] M. Madueño Ortega, et al. (2013). *Control teleoperado del robot RV-M1 mediante dispositivo móvil y Realidad Aumentada*. [Online] [Consulta: 30 de mayo de 2015] http://upcommons.upc.edu/pfc/bitstream/2099.1/19144/2/Resum.pdf

[24] R. Martínez. (2011). *Tecnologías hápticas para mejorar la seguridad en la conducción*. [Online] [Consulta: 15 de junio de 2015] https://riunet.upv.es/handle/10251/38809

[25] R. Harris. (2005). *Voice interaction design: crafting the new conversational speech systems*. [Online] [Consulta: 30 de mayo de 2015] www.sciencedirect.com/science/book/9781558607682

[26] A. Widmer. (2014). *Using google glass to enhance pre-hospital care* [Online] [Consulta: 15 de junio de 2015] www.medical-informatics.ch/index.php/smiojs/article/view/316/358

[27] DARPA Grand Challenge. [Online] [Consulta: 20 de Mayo de 2015] http://archive.darpa.mil/grandchallenge04/overview.htm

[28] R. Behringer, et al. (2004). *Development of an autonomous vehicle for the DARPA Grand Challenge.* [Online] [Consulta: 20 de mayo de 2015] http://www.gavlab.auburn.edu/uploads/Development%20DARPA%20Grand%20Challenge.pdf

[29] DARPA Urban Challenge. [Online] [Consulta: 20 de mayo de 2015] http://archive.darpa.mil/grandchallenge/overview.htm

[30] R.M. Murray. "Autonomous Machines: Racing to Win the DARPA Grand Challenge". Proc. 2005 *American Control Conference.,* Oregon, pp. 9-10.

[31] M. Davidow. (2006). *Stanley: The robot that won the DARPA Grand Challenge.* [Online] [Consulta: 20 de mayo de 2015] http://robots.stanford.edu/papers/thrun.stanley05.pdf

[32] P. Grey. (2012). *Bumblebee: stereo vision camera systems.* [Online] [Consulta: 15 de junio de 2015] http://www.docslides.com/briana-ranney/bumblebee-stereo-vision-camera

[33] S. S. Intelligence. (2006). "Lms200/211/221/291 laser measurement systems," Technical Description. [Online] [Consulta: 30 de mayo de 2015]http://www.sick.com/group/EN/home/about_sick/journals/Documents/1_2006_e.pdf

[34] S. Cousins, B. Gerkey, K. Conley, and Willow Garage. (2010). *Sharing Software with ROS.* [Online] [Consulta: 20 de mayo de 2015] http://ieeexplore.ieee.org.strauss.uc3m.es:8080/stamp/stamp.jsp?tp=&arnumber=5480439

[35] W. Hongxing; H. Zhen; Y. Qiang; L. Miao; G. Yong; T. Jindong. (2014). *RGMP-ROS: A real-time ROS architecture of hybrid RTOS and GPOS on multi-core processor Robotics and Automation (ICRA).* [Online] [Consulta: 15 de junio de 2015] http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?reload=true&arnumber=6907205

[36] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. (2009). *ROS: an open-source Robot Operating System.* [Online] [Consulta: 30 de mayo de 2015] https://www.willowgarage.com / sites/default/files/icraoss09-ROS.pdf

[37] Qt creator. [Online] [Consulta: 30 de mayo de 2015]: http://wiki.qt.io/About_Qt

[38] CMakeLists. [Online] [Consulta: 30 de mayo de 2015] http://wiki.ros.org/catkin/CMakeLists.txt

[39] Catkin. [Online] [Consulta: 30 de mayo de 2015] http://wiki.ros.org/catkin/package.xml

[40] Qt-custom-gauge-widget. [Online] [Consulta: 30 de mayo de 2015] https://github.com/Berrima/Qt-custom-gauge-widget/tree/master/source

[41] Qt-circular-gauges. [Online] [Consulta: 30 de mayo de 2015] http://pytricity.com/qt-circular-gauges/

[42] Hp Pavilion Dv6-3183es. (2011) [Online] [Consulta: 30 de mayo de 2015] http://h20564.www2.hp.com/hpsc/doc/public/display?docId=c02754741

[43] ROS en Ubuntu. [Online] [Consulta: 30 de mayo de 2015] http://wiki.ros.org/indigo/Installation/Ubuntu

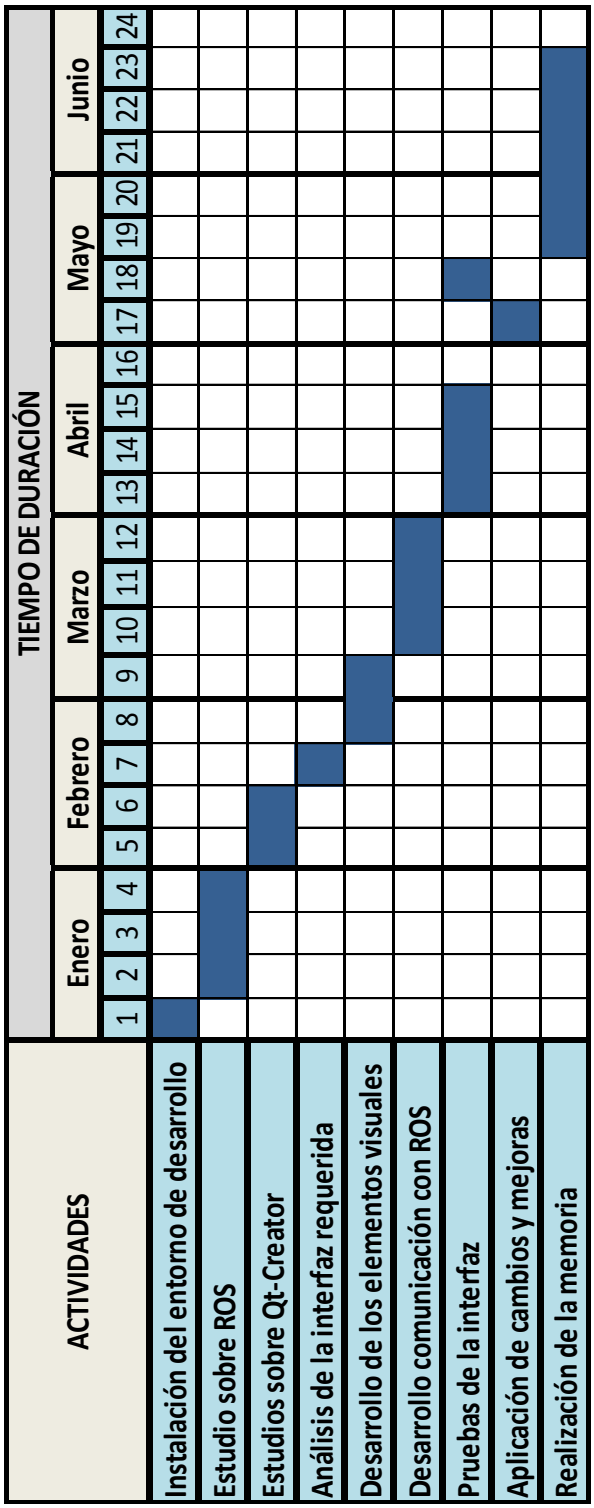
[44] Repositorios de Ubuntu. [Online] [Consulta: 30 de mayo de 2015] https://help.ubuntu.com/community/Repositories/Ubuntu

INDICE DE TABLAS Y FIGURAS

FIGURAS

TABLAS

Anexo I: Diagrama de Gantt del proyecto



Anexo II: Instalación del SO Ubuntu 14.04

Como este portátil disponía de un sistema operativo Windows 7, se llevó a cabo la instalación de Ubuntu 14.04 en una partición del disco duro del portátil. Para realizar la partición primero hay que acceder a "Administración de equipos", desde donde efectuaremos una liberación de espacio en el disco para efectuar posteriormente la instalación de Ubuntu en esta partición.

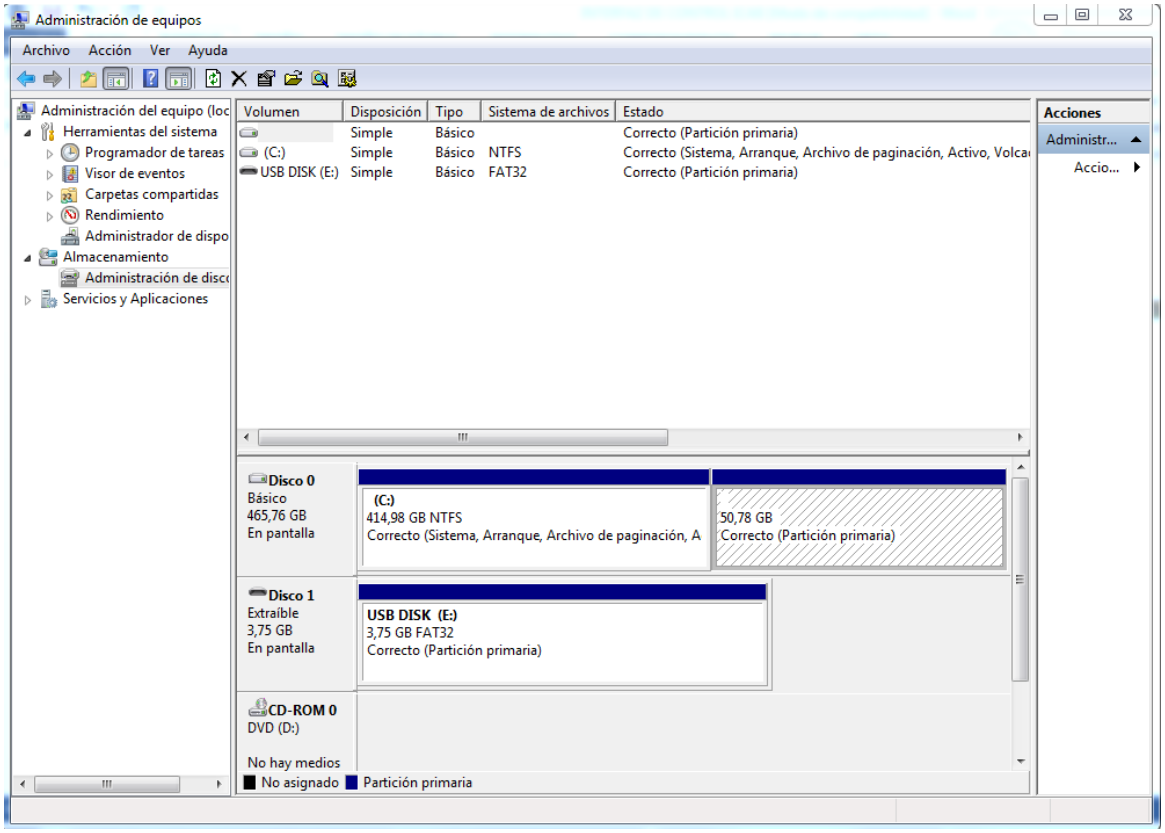


Figura 1: *Administración de equipos Windows 7.*

La partición efectuada es de 50,78Gb, dónde el sistema operativo de Ubuntu puede funcionar sin ningún problema, ya que es bastante ligero.

Una vez realizada la partición es necesario descargar el sistema operativo de la página oficial de Ubuntu http://www.ubuntu.com/download/desktop, en este caso se escogió la versión recomendada 14.04.2, ya que es la que mayor soporte ofrece. Una vez descargado, fue necesario generar un USB de arranque (bootable), que es equivalente al CD de arranque. Para ello se utilizó el programa "LinuxLive USB Creator" descargado de la página oficial http://www.linuxliveusb.com/, una vez instalado, se lanzó el programa y se

procedió a la creación del USB, que debe tener al menos 2Gb, ya que el SO ocupa 1,3Gb. Para proceder a la instalación es necesario reiniciar el ordenador y arrancar el ordenador desde el USB, iniciándose el CD Live de Ubuntu en vez de nuestro sistema operativo original.

Este CD Live arrancara mostrando el escritorio del sistema operativo de Ubuntu, en el que se encontrara el programa de instalación "Instalar Ubuntu 14.04 LTS", a través del que se realiza la instalación.

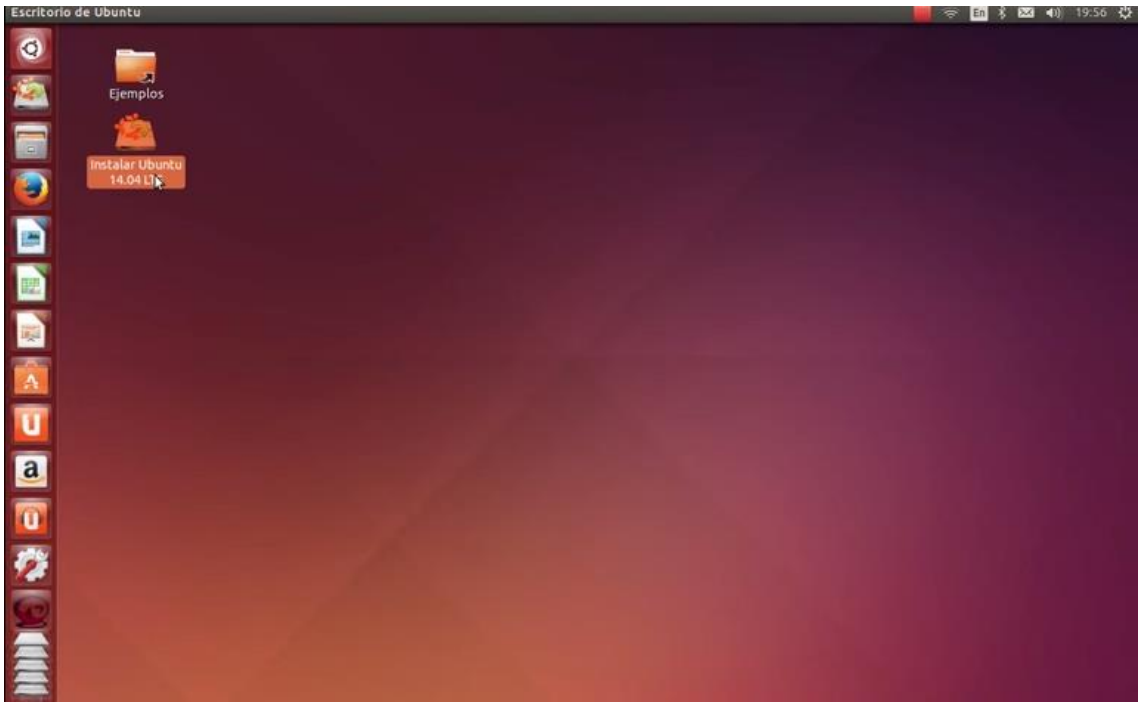


Figura 9. 2: *Aplicación de instalación ubicada en el escritorio.*

En la instalación se seleccionara el idioma, realizaremos la instalación sin conexión a internet, y se selecciona el tipo de instalación en la figura 4.2, escogiendo la opcion "install alongside windows 7" y se cpontinua con la instalación.

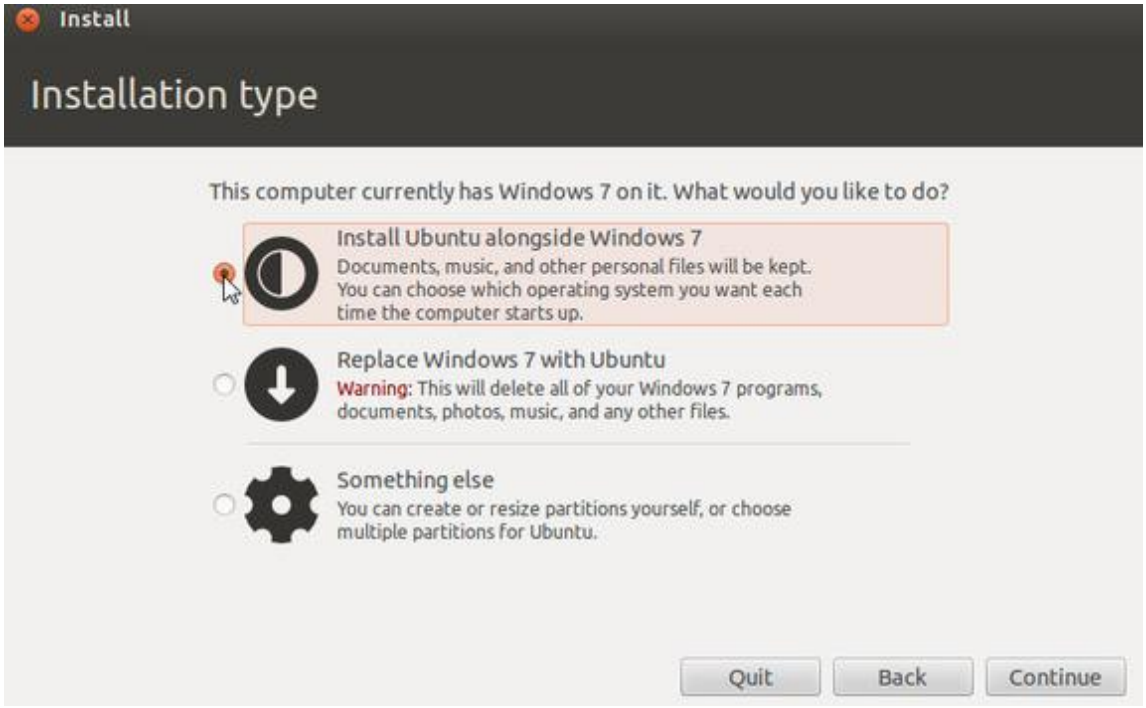


Figura 9. 3: *Menú de selección del tipo de instalación deseada.*

Posteriormente solicitará la partición sobre la que se desea instalar el sistema operativo, eligiendo la partición creada previamente, y se iniciará la instalación de Ubuntu 14.04. Al finalizar solicitará que se introduzca un nombre de usuario y una contraseña, y después hay que reiniciar el sistema arrancando ya desde el disco duro, que nos mostrará el menú de arranque:

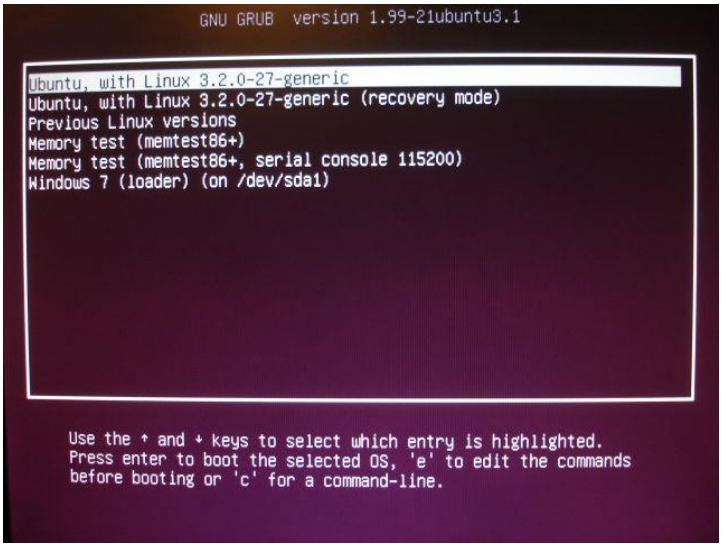


Figura 9. 4: *Arranque del dual Boot de Ubuntu con Windows.*

Seleccionamos el primero de la lista, disponiendo de ambos sistemas, para proceder a continuación a la instalación y manejo de ROS dentro del SO Ubuntu.

Anexo III: Instalación y configuración inicial de ROS

Lo primero que se desarrolló antes de trabajar sobre la interfaz fue el estudio del funcionamiento de ROS y las partes que lo componían.

Para llevar a cabo el aprendizaje sobre ROS hay una página con un tutorial completo sobre las partes que lo componen http://wiki.ros.org/ROS/Tutorials. A continuación se explicaran los conceptos generales de ROS.

Previamente a la ejecución del tutorial es necesario llevar a cabo la instalación de ROS, que en nuestro caso se instaló el paquete ROS Indigo, que es la penúltima versión actualmente, pero que cuando se inició el proyecto era la más reciente. Para instalarlo hay que seguir los siguientes pasos a través de la "terminal" de Ubuntu [43].

Configurar los repositorios de Ubuntu, que se encuentran divididos en 4 áreas: **Main** (soporte oficial), **Restricted (**soporte oficial, pero no bajo una licencia libre completa), **Universe** (mantenido por la comunidad) y **Multiverse** (*software* de pago) [44]. Hay que activar todas las áreas en el "Ubuntu *Software* Tab", mostrado a continuación. [39]

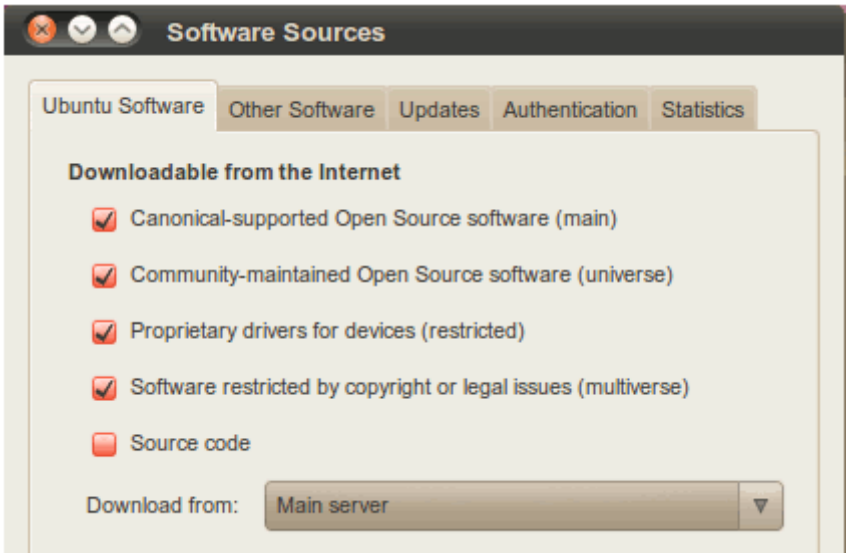


Figura 9. 5: *Menú para habilitar repositorios de Ubuntu.*

Configurar el archivo "sources.list" para aceptar los paquetes provenientes de ROS mediante la siguiente línea de comando:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Lo que realiza es la escritura 'echo' de "deb http://..." en el archivo "ros-latest.list" dentro del directorio del "sources.list.d", además hay que configurar las claves para el acceso al repositorio mediante la línea de comando:

```
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net --recv-key 0xB01FA116
```

1) Actualizamos el repositorio ejecutando el comando "**sudo apt-get update**", se inspeccionan los paquetes disponibles "**apt-cache search ros-indigo**", y ejecutamos el comando de instalación del paquete completo.

```
sudo apt-get install ros-indigo-desktop-full
```

2) Una vez instalado antes de proceder al uso de ROS se inicializará la herramienta rosdep ("**sudo rosdep init**"), que permite instalar dependencias adicionales de ROS a la hora de compilar paquetes cuando estos lo requieran. Una vez inicializado se actualizará con el comando "**rosdep update**".

3) Configurar el archivo "**.bashrc**" que configura el inicio de sesión de la terminal, de modo que si se incluye la operación para actualizar las variables de entorno de ROS cada vez que se abra la ventana, se habran actualizado, evitando realizar esta operación manualmente. Para ello se edita el archivo "**.bashrc**" añadiendo la línea "**source /opt/ros/indigo/setup.bash**" al final del texto.

4) Instalar rosinstall, que sirve para descargar paquetes de ROS, para ello se ejecuta el comando:

```
sudo apt-get install python-rosinstall
```

Ahora que ya está instalado ROS, se empieza a ejecutar el tutorial, que tiene como primer punto la configuración del "workspace" (espacio de trabajo), sobre el que se trabaja a lo largo del proyecto. Lo primero es la creación de las carpetas del directorio del workspace, situado en la carpeta personal del sistema se crea una carpeta con el nombre de "catkin_ws" (catkin workspace), y otra carpeta dentro de esta llamada "src" (source), donde se incluirán los paquetes sobre los que se trabaja, y por último se inicializa este espacio a través de dos comandos:

```
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

El primero sirve para situarse dentro de la carpeta creada, y el segundo es el de indicarle a ROS que este es el directorio de trabajo de catkin. Ahora se puede ejecutar desde el directorio de "/catkin_ws" el comando "catkin_make", que construye los paquetes generando una carpeta "devel" y otra "build". En la primera se incorporara un archivo "setup.bash" que sirve para añadir los paquetes compilados al espacio de trabajo de ROS. Si ejecutamos este archivo con el comando "source" añadirá a la variable global ROS_PACKAGE_PATH el directorio de catkin (home/user_name/catkin_ws/src).

Anexo IV: CMakeList.txt

```
#############################################################
# CMake
#############################################################

cmake_minimum_required(VERSION 2.8.0)
project(icab_reconfigure)
set(CMAKE_BUILD_TYPE Debug)

#############################################################
# Catkin
#############################################################

find_package(catkin REQUIRED COMPONENTS qt_build roscpp)
include_directories(${catkin_INCLUDE_DIRS})

catkin_package()

#############################################################
# Qt Environment
#############################################################

# this comes from qt_build's qt-ros.cmake which is automatically
# included via the dependency call in package.xml
rosbuild_prepare_qt4(QtCore QtGui) # Add the appropriate
components to the component list here

#############################################################
# Sections
#############################################################

file(GLOB QT_FORMS RELATIVE ${CMAKE_CURRENT_SOURCE_DIR} ui/*.ui)
file(GLOB QT_RESOURCES RELATIVE ${CMAKE_CURRENT_SOURCE_DIR}
resources/*.qrc)
file(GLOB_RECURSE QT_MOC RELATIVE ${CMAKE_CURRENT_SOURCE_DIR}
FOLLOW_SYMLINKS include/icab_reconfigure/*.hpp)

QT4_ADD_RESOURCES(QT_RESOURCES_CPP ${QT_RESOURCES})
QT4_WRAP_UI(QT_FORMS_HPP ${QT_FORMS})
QT4_WRAP_CPP(QT_MOC_HPP ${QT_MOC})

#############################################################
# Sources
#############################################################

File (GLOB_RECURSE QT_SOURCES RELATIVE
${CMAKE_CURRENT_SOURCE_DIR} FOLLOW_SYMLINKS src/*.cpp)

#############################################################
# Binaries
#############################################################

add_executable (icab_reconfigure ${QT_SOURCES}
${QT_RESOURCES_CPP} ${QT_FORMS_HPP} ${QT_MOC_HPP})
```

```
target_link_libraries (icab_reconfigure ${QT_LIBRARIES} ${catkin_LIBRARIES})
install(TARGETS icab_reconfigure RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

Anexo V: Package.xml

```
<?xml version="1.0"?>
<package>
  <name>icab_reconfigure</name>
  <version>0.1.0</version>
  <description>
    icab_reconfigure
  </description>
  <maintainer email="lsi-ros@gmail.com">lsi-ros</maintainer>
  <author>lsi-ros</author>
  <license>BSD</license>

  <!--  <url  type="bugtracker">  https://github.com/stonier/qt_ros/issues
</url> -->
  <!-- <url type="repository">https://github.com/stonier/qt_ros</url> -->


  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>qt_build</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>libqt4-dev</build_depend>
  <build_depend>icab_msgs</build_depend>
  <run_depend>qt_build</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>libqt4-dev</run_depend>
  <run_depend>icab_msgs</run_depend>

</package>
```